API Samples

Assemblies

Analysis

Interference Analysis

Bill of Materials

Using the BOM APIs
Exporting the assembly BOM

Constraints

Add assembly insert constraint
Add assembly mate constraint
Add mate constraint using work planes in parts
Add mate constraint with limits
Create planar AssemblyJoint with offset to origins
Create rotational assembly joint
Add iMate Definition
iMate Result Creation

General

Add assembly occurrences to a new folder
Demote occurence
Promote occurence
Assembly Ground Occurrences
Create Revit Export sample
Open assembly using last model state
Create a model state
Associative body copy
Shrink wrap substitute in assembly

Occurrences

Adding iAssembly occurrences
Adding iPart occurrences to an assembly
Assembly Add Occurrence
iMate Creation During Occurrence Placement
Traverse an Assembly
Create assembly occurrence with representations
Print instance properties of all components in an assembly
Assembly Move Occurrence

B-Rep and Geometry

Transient Modeling

Create Approximate Polyline to 3D Curve
Extract a Partial Curve from a Curve
Body Imprinting and matching the results
Imprint bodies within an assembly.
3D Curve from Parametric Curve
Split a 2D Curve
Transient solid body creation
Tapered Surface Using Offset Curve and Ruled Surface
Client graphics creation of 3D primitives
Transient B-Rep Ruled Surface with Lines
Transient B-Rep Ruled Surface with Arc and Line

BIM Exchange

General

Export to IFC Format Sample

Content Center

General

Replace content center part
Place Content Center Parts

Design Simulation

General

Play back a simulation

Drawings

Annotations

Center Dimension Text
Baseline dimension sets
Create bend note
Chain dimensions sets
Bullet and Numbering List
Drawing Welding Symbol Creation
EdgeSymbol Creation Sample
Create a feature control frame symbol
Add a general note
Creating Stacked Text
Create thread note
Add new leader note
Create ordinate dimension
create punch note
RevisionCloud Creation Sample
Query revision table
Create SketchedSymbol Definition
Create sketched symbol and leader
Add surface texture symbol to dimension
Title Block Definition Create and Insert
Copying a title block definition

AutoCAD

AutoCAD block definitions import
AutoCAD block insertion

Balloons

Balloons - edit
Creation a balloon
Find component referenced by balloon

Dimensions

Create linear foreshortened dimension sample
Dimensions - edit
Find and remove unattached dimensions
Aligning drawing dimensions

General

Border Create and Insert
Printing - All Sheets in Drawing
Drawing Sketch Hatch Region Sample
Adding and editing a feature control frame
Border Insert
Border Insert Default
Sketched Symbol Definition Library Creation

Tables and Lists

Custom Table - create
Create a Bend Table
Create a Configuration Table
Create a drawing Excel Table
Creating hole tables

Creating a parts list
Parts List Edit
Parts List Query

Views

Creation of a break operation in a drawing view
Create BreakOpertion by Sketch Sample
CropOperation creation sample
Drawing Sketches - editing line type and color
Sketch fill region
Sketch within a drawing view
Adding Representation views
Create flat pattern drawing view
Create base view with representations
Add detail drawing view
Draft Views - create
Moving sketch entities to a new layer
Move DrawingViewAnnotation Text Position Sample
Break alignment of a section view
Create sheet with multiple views

General

Client Graphics

Client Graphics - Draw Range Box
Client graphics texture-based color mapping
Client Graphics - Vertex Color by Z Height
Client Graphics - Line
Client graphics from SAT file body
Text Using Client Graphics (Simple)
Text Using Client Graphics (Multiple fonts and lines)
Client Graphics - Triangle
Anchored Client Grahics
Client graphics - image in point graphics
Selection of Surface Graphics Primitives
Create curve primitives

iProperties

Update iProperty values using Apprentice
Create custom iProperties
Create or update custom iProperty
Get value of iProperty

Miscellaneous

Printing - Drawing Print
Drive the camera
Pack and go
Printing - Part or Assembly
Browser Search Box Sample
Show help sample with local help
Show online help sample for file dialog
Inventor theme change sample
UCS by three points
UCS by transformation matrix

Parameters

Display information about parameter tolerances.
Create user parameters
Set the value of a parameter
Text and Boolean user parameter creation

Projects

Set active project
Create project
Query and create library paths

Materials and Appearances

Appearances

Change color of part, features or faces
Create a simple appearance.
Write out all appearance information to a file.
Write out all document appearances
RemoveAssemblyOverrides
Removes all appearance overrides in a part.
Set the appearance of an occurrence.

Materials

Write out all materials to a file.
Write out all physical properties to a file.
Write out all document materials to a file.
Write out all document physical properties to a file.

Parts

Analysis

Create a Draft Analysis

2D Sketches

Control point, equation, and intersection curve creation.
Creates an Arc Length Dimension Constraint
Create Centerpoint Rectangles
Copy a sketch
ImportedDWGComponent Creation
Sketch from Face Silhouette
Sketch Edit Orientation
Sketch Share
Sketch Add
Sketch Add Oriented
Querying a sketch profile to get regions.
Sketch profile control
Projection - project across parts
Copy sketch contents
Defer sketch updates
Sketch Delete
Sketch Open for Edit
Move sketch entities
Offset a 2D sketch
Sketch Lines
Set Sketch Visibility
Create and insert a sketch block definition into a part sketch
Create sketch block from an existing sketch
Create sketch elliptical arc
Sketch Display Entities
Spline - create NURBS
Create slots in sketch.
Sketch Spline
Sketch Text Add

3D Sketches

Create a 3D sketch dimension
OnFaceCurve creation

Features

Delete Face, Boundary Patch and Stitch features
SurfaceBody Copy
Create Pattern Feature with PatternBoundarySetting Sample
Add a decal feature
Display feature information
Extrude Feature - Create Block with Pocket
Edit profile of an extrude feature
Extrude sketch text
Fillet Feature (All Rounds)
Fillet Feature (Complex)
Fillet Feature (Simple)
Finish Feature Creation

Highlight Feature Faces
Hole Feature - Through holes (RegularAndTapped)
Hole feature linear placement
Placement of a standard iFeature
Place table driven iFeature
Changing row of table driven iFeature
Adding a new stitch (knit) feature
Loft Feature With Non-Planar Section
Mark feature creation sample
Move Feature Creation
Create and Edit an Extrude Feature with a pocket
Partial Chamfer Sample
Part SimplifyFeature Sample
Sweep Feature Add
Thread Feature Create
Edit thread features
Creating a ThreadInfo object

General

Creating a new parameter group
Derived Parts and Assemblies
Selectively link paramaters
Query feature dimensions
Associatively import AutoCAD
Basic Selection Using Interaction Events
Is cylindrical face interior or exterior?
Mass Properties from Part
Computing mass properties without dirtying document
Work point at mass center
Modify Multiple Model States Sample
Model Parameters
Table Parameters
Transient surface body creation
Create primitive BRep

Sheet Metal

Edit width extent of an existing flange feature
Creating flange features
Finding Bend Extent (Tangent) Edges
Create sheet metal face and fold features
Create sheet metal lofted flange feature
Report on punch feature ID's
Add a punch tool feature
Create sheet metal rip feature
Sheet Metal Feature Display
Create sheet metal face and cut features
Create sheet metal face and flange features
Sheet Metal Style Display
Sheet Metal Thickness Editing
Sheet Metal Style Creation
Translate - Sheet Metal to DXF

Translators

Export

Publish FlatPattern to SAT
Publish FlatPattern to STEP
Export to 3D PDF
Export to USDz
Publish FlatPattern to DXF
Save as DWF Translator Sample
Save as DWG Translator Sample
Save as DXF Translator Sample
Save as IGES Translator Sample
Save as PDF Translator Sample
Save as STEP Translator Sample
Export to DWF
Export to DWG
Export to DXF
Export to IGES

Export to STEP
Export to PDF

Import

Open a Catia file using the Catia Translator Sample
Import DWG into sketch
Open an NX file suing the NX Translator Sample
Import Revit data into Inventor
Open Rhino Translator Sample
Open an STL file using the STL Translator Sample

User Interaction

Custom Commands

InteractionGraphics

General

Creating a HighlightSet
Using measure events
Post Private Event Sample
Prompt message creation sample
WebBrowserDialog creation

User Interface

Add commands to the application menu
Adjust the brightness of lighting
Navigation between browser and data
Single selection - simple
Add parallel environment with contextual tabs
Dockable window
Dock browser pane to a custom ViewFrame
Print list of all Inventor Commands
Print information about all available ribbons.
Creation of an override environment for a document
Using Inventor's error dialog
Display custom error messages
File Dialog
MiniToolbarsample
Using Inventor's progress bars
Create a ribbon panel in an existing tab
Window Selection
Cancel a double click
OnDrag Event - dragging a WorkPoint
Move view tab between different view frames

Interference Analysis

Description

This sample demonstrates the functions used to calculate interference analysis in an assembly.

Code Samples

- VBA
- iLogic

To use this sample have an assembly open that contains mutiple parts. Depending on preselected parts when running the sample, you'll get different results. If one part is selected, that one part will be checked against the rest of the assembly. If more than one part is selected you have the choice of checking for interference among the selected parts or checking the selected parts against the rest of the assembly. If no parts are selected it will check every part against every other part.

Copy Code

```
Public Sub Interference()
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Find all selected occurrences and add them to an ObjectCollection.
    Dim oSelectedOccs As ObjectCollection
    Set oSelectedOccs = ThisApplication.TransientObjects.CreateObjectCollection
    Dim i As Long
    For i = 1 To oDoc.SelectSet.Count
        If oDoc.SelectSet.Item(i).Type = kComponentOccurrenceObject Then
            oSelectedOccs.Add oDoc.SelectSet.Item(i)
```

```
        End If
    Next

    ' If no occurrences are selected check for interference of
    ' all parts against all parts.  If one occurrence is selected, check
    ' for interference between that occurrence and the rest of the assembly.
    ' If more than one occurrence is selected let the user decide if it
    ' should check for interference between the parts in the selection or
    ' between the selected parts and the rest of the assembly.
    Dim oResults As InterferenceResults
    Dim oCheckSet As ObjectCollection
    Set oCheckSet = ThisApplication.TransientObjects.CreateObjectCollection
    If oSelectedOccs.Count = 0 Then
        ' Add all occurrences to the object collection
        Dim oOcc As ComponentOccurrence
        For Each oOcc In oDoc.ComponentDefinition.Occurrences
            oCheckSet.Add oOcc
        Next

        ' Get the interference between everything.
        Set oResults = oDoc.ComponentDefinition.AnalyzeInterference(oCheckSet)
    ElseIf oSelectedOccs.Count = 1 Then
        ' Add all occurrences except the selected occurrence to the object collection.
        For Each oOcc In oDoc.ComponentDefinition.Occurrences
            If Not oOcc Is oSelectedOccs.Item(1) Then
                oCheckSet.Add oOcc
            End If
        Next

        ' Get the interference between the selected occurrence everything else.
        Set oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs, oCheckSet)
    Else
        If MsgBox("Check interference between selected occurrences and all other occurrences?", vbYesNo + vbQuestion) = vbYes Then
            ' Add all occurrences except the selected occurrences to the object collection.
            For Each oOcc In oDoc.ComponentDefinition.Occurrences
                ' Check to see if this occurrences is already selected.
                Dim bSelected As Boolean
                bSelected = False
                For i = 1 To oSelectedOccs.Count
                    If oSelectedOccs.Item(i) Is oOcc Then
                        bSelected = True
                        Exit For
                    End If
                Next

                If Not bSelected Then
                    oCheckSet.Add oOcc
                End If
            Next

            ' Check interference between the selected items.
            Set oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs, oCheckSet)
        Else
            ' Check interference between the selected items.
            Set oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs)
        End If
    End If

    If oResults.Count = 1 Then
        MsgBox "There is 1 interference."
    ElseIf oResults.Count > 1 Then
        MsgBox "There are " & oResults.Count & " interferences."
    End If

    If oResults.Count > 0 Then
        Dim oHS1 As HighlightSet
        Set oHS1 = oDoc.HighlightSets.Add
        oHS1.Color = ThisApplication.TransientObjects.CreateColor(255, 0, 0)
        Dim oHS2 As HighlightSet
        Set oHS2 = oDoc.HighlightSets.Add
        oHS2.Color = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

        For i = 1 To oResults.Count
            oHS1.Clear
            oHS2.Clear
            oHS1.AddItem oResults.Item(i).OccurrenceOne
            oHS2.AddItem oResults.Item(i).OccurrenceTwo
            MsgBox "Occurrences are highlighted from interference " & i
        Next

        oHS1.Clear
        oHS2.Clear
    Else
        MsgBox "There is no interference."
    End If
End Sub
```

To use this sample have an assembly open that contains mutiple parts. Depending on preselected parts when running the sample, you'll get different results. If one part is selected, that one part will be checked against the rest of the assembly. If more than one part is selected you have the choice of checking for interference among the selected parts or checking the selected parts against the rest of the assembly. If no parts are selected it will check every part against every other part.

Copy Code

```
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    ' Find all selected occurrences and add them to an ObjectCollection.
    Dim oSelectedOccs As ObjectCollection
    oSelectedOccs = ThisApplication.TransientObjects.CreateObjectCollection
    Dim i As Long
    For i = 1 To oDoc.SelectSet.Count
        If oDoc.SelectSet.Item(i).Type = kComponentOccurrenceObject Then
            oSelectedOccs.Add(oDoc.SelectSet.Item(i))
        End If
    Next

    ' If no occurrences are selected check for interference of
```

```
    ' all parts against all parts.  If one occurrence is selected, check
    ' for interference between that occurrence and the rest of the assembly.
    ' If more than one occurrence is selected let the user decide if it
    ' should check for interference between the parts in the selection or
    ' between the selected parts and the rest of the assembly.
    Dim oResults As InterferenceResults
    Dim oCheckSet As ObjectCollection
    oCheckSet = ThisApplication.TransientObjects.CreateObjectCollection
    If oSelectedOccs.Count = 0 Then
        ' Add all occurrences to the object collection
        Dim oOcc As ComponentOccurrence
        For Each oOcc In oDoc.ComponentDefinition.Occurrences
            oCheckSet.Add(oOcc)
        Next

        ' Get the interference between everything.
        oResults = oDoc.ComponentDefinition.AnalyzeInterference(oCheckSet)
    ElseIf oSelectedOccs.Count = 1 Then
        ' Add all occurrences except the selected occurrence to the object collection.
        For Each oOcc In oDoc.ComponentDefinition.Occurrences
            If Not oOcc Is oSelectedOccs.Item(1) Then
                oCheckSet.Add(oOcc)
            End If
        Next

        ' Get the interference between the selected occurrence everything else.
        oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs, oCheckSet)
    Else
        If MsgBox("Check interference between selected occurrences and all other occurrences?", vbYesNo + vbQuestion) = vbYes Then
            ' Add all occurrences except the selected occurrences to the object collection.
            For Each oOcc In oDoc.ComponentDefinition.Occurrences
                ' Check to see if this occurrences is already selected.
                Dim bSelected As Boolean
                bSelected = False
                For i = 1 To oSelectedOccs.Count
                    If oSelectedOccs.Item(i) Is oOcc Then
                        bSelected = True
                        Exit For
                    End If
                Next

                If Not bSelected Then
                    oCheckSet.Add(oOcc)
                End If
            Next

            ' Check interference between the selected items.
            oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs, oCheckSet)
        Else
            ' Check interference between the selected items.
            oResults = oDoc.ComponentDefinition.AnalyzeInterference(oSelectedOccs)
        End If
    End If

    If oResults.Count = 1 Then
        MsgBox("There is 1 interference.")
    ElseIf oResults.Count > 1 Then
        MsgBox("There are " & oResults.Count & " interferences.")
    End If

    If oResults.Count > 0 Then
        Dim oHS1 As HighlightSet
        oHS1 = oDoc.HighlightSets.Add
        oHS1.Color = ThisApplication.TransientObjects.CreateColor(255, 0, 0)
        Dim oHS2 As HighlightSet
        oHS2 = oDoc.HighlightSets.Add
        oHS2.Color = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

        For i = 1 To oResults.Count
            oHS1.Clear
            oHS2.Clear
            oHS1.AddItem(oResults.Item(i).OccurrenceOne)
            oHS2.AddItem(oResults.Item(i).OccurrenceTwo)
            MsgBox("Occurrences are highlighted from interference " & i)
        Next

        oHS1.Clear
        oHS2.Clear
    Else
        MsgBox("There is no interference.")
    End If
```

Using the BOM APIs

Description

This sample demonstrates the Bill of Materials API functionality in assemblies.

Code Samples

- VBA
- iLogic

Have an assembly document open and run the following sample.

Copy Code

```
Public Sub BOMQuery()
    ' Set a reference to the assembly document.
```

```
    ' This assumes an assembly document is active.
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim FirstLevelOnly As Boolean
    If MsgBox("First level only?", vbYesNo) = vbYes Then
        FirstLevelOnly = True
    Else
        FirstLevelOnly = False
    End If

    ' Set a reference to the BOM
    Dim oBOM As BOM
    Set oBOM = oDoc.ComponentDefinition.BOM

    ' Set whether first level only or all levels.
    If FirstLevelOnly Then
        oBOM.StructuredViewFirstLevelOnly = True
    Else
        oBOM.StructuredViewFirstLevelOnly = False
    End If

    ' Make sure that the structured view is enabled.
    oBOM.StructuredViewEnabled = True

    'Set a reference to the "Structured" BOMView
    Dim oBOMView As BOMView
    Set oBOMView = oBOM.BOMViews.Item("Structured")

    Debug.Print "Item"; Tab(15); "Quantity"; Tab(30); "Part Number"; Tab(70); "Description"
    Debug.Print "--------------------------------------------------------------------------------"

    'Initialize the tab for ItemNumber
    Dim ItemTab As Long
    ItemTab = -3
    Call QueryBOMRowProperties(oBOMView.BOMRows, ItemTab)
End Sub

Private Sub QueryBOMRowProperties(oBOMRows As BOMRowsEnumerator, ItemTab As Long)
    ItemTab = ItemTab + 3
    ' Iterate through the contents of the BOM Rows.
    Dim i As Long
    For i = 1 To oBOMRows.Count
        ' Get the current row.
        Dim oRow As BOMRow
        Set oRow = oBOMRows.Item(i)

        'Set a reference to the primary ComponentDefinition of the row
        Dim oCompDef As ComponentDefinition
        Set oCompDef = oRow.ComponentDefinitions.Item(1)

        Dim oPartNumProperty As Property
        Dim oDescripProperty As Property

        If Typeof oCompDef Is VirtualComponentDefinition Then
            'Get the file property that contains the "Part Number"
            'The file property is obtained from the virtual component definition
            Set oPartNumProperty = oCompDef.PropertySets _
                .Item("Design Tracking Properties").Item("Part Number")

            'Get the file property that contains the "Description"
            Set oDescripProperty = oCompDef.PropertySets _
                .Item("Design Tracking Properties").Item("Description")

            Debug.Print Tab(ItemTab); oRow.ItemNumber; Tab(17); oRow.ItemQuantity; Tab(30); _
                oPartNumProperty.Value; Tab(70); oDescripProperty.Value
        Else
            'Get the file property that contains the "Part Number"
            'The file property is obtained from the parent
            'document of the associated ComponentDefinition.
            Set oPartNumProperty = oCompDef.Document.PropertySets _
                .Item("Design Tracking Properties").Item("Part Number")

            'Get the file property that contains the "Description"
            Set oDescripProperty = oCompDef.Document.PropertySets _
                .Item("Design Tracking Properties").Item("Description")

            Debug.Print Tab(ItemTab); oRow.ItemNumber; Tab(17); oRow.ItemQuantity; Tab(30); _
                oPartNumProperty.Value; Tab(70); oDescripProperty.Value

            'Recursively iterate child rows if present.
            If Not oRow.ChildRows Is Nothing Then
                Call QueryBOMRowProperties(oRow.ChildRows, ItemTab)
            End If
        End If
    Next
    ItemTab = ItemTab - 3
End Sub
```

Have an assembly document open and run the following sample.

Copy Code

```
Sub Main
    ' Set a reference to the assembly document.
    ' This assumes an assembly document is active.
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim FirstLevelOnly As Boolean
    If MsgBox("First level only?", vbYesNo) = vbYes Then
        FirstLevelOnly = True
    Else
        FirstLevelOnly = False
    End If

    ' Set a reference to the BOM
    Dim oBOM As BOM
```

```
    oBOM = oDoc.ComponentDefinition.BOM

    ' Set whether first level only or all levels.
    If FirstLevelOnly Then
        oBOM.StructuredViewFirstLevelOnly = True
    Else
        oBOM.StructuredViewFirstLevelOnly = False
    End If

    ' Make sure that the structured view is enabled.
    oBOM.StructuredViewEnabled = True

    'Set a reference to the "Structured" BOMView
    Dim oBOMView As BOMView
    oBOMView = oBOM.BOMViews.Item("Structured")

    Logger.Info("Item".PadRight(15) + "Quantity".PadRight(15) + "Part Number".PadRight(40) + "Description")

    Logger.Info("--------------------------------------------------------------------------------")


    'Initialize the tab for ItemNumber
    Dim ItemTab As Long
    ItemTab = -3
    Call QueryBOMRowProperties(oBOMView.BOMRows, ItemTab)
End Sub

Private Sub QueryBOMRowProperties(oBOMRows As BOMRowsEnumerator, ItemTab As Long)
    ItemTab = ItemTab + 3
    ' Iterate through the contents of the BOM Rows.
    Dim i As Long
    For i = 1 To oBOMRows.Count
        ' Get the current row.
        Dim oRow As BOMRow
        oRow = oBOMRows.Item(i)

        'Set a reference to the primary ComponentDefinition of the row
        Dim oCompDef As ComponentDefinition
        oCompDef = oRow.ComponentDefinitions.Item(1)

        Dim oPartNumProperty As Inventor.Property
        Dim oDescripProperty As Inventor.Property

        If Typeof oCompDef Is VirtualComponentDefinition Then
            'Get the file property that contains the "Part Number"
            'The file property is obtained from the virtual component definition
            oPartNumProperty = oCompDef.PropertySets _
                .Item("Design Tracking Properties").Item("Part Number")

            'Get the file property that contains the "Description"
            oDescripProperty = oCompDef.PropertySets _
                .Item("Design Tracking Properties").Item("Description")

            Logger.Info("".PadRight(ItemTab) + oRow.ItemNumber.PadRight(Max(15-ItemTab,0)) + CStr(oRow.ItemQuantity).PadRight(15) + oP
        Else
            'Get the file property that contains the "Part Number"
            'The file property is obtained from the parent
            'document of the associated ComponentDefinition.
            oPartNumProperty = oCompDef.Document.PropertySets _
                .Item("Design Tracking Properties").Item("Part Number")

            'Get the file property that contains the "Description"
            oDescripProperty = oCompDef.Document.PropertySets _
                .Item("Design Tracking Properties").Item("Description")

            Logger.Info("".PadRight(ItemTab) + oRow.ItemNumber.PadRight(Max(15-ItemTab, 0)) + CStr(oRow.ItemQuantity).PadRight(15) + o

            'Recursively iterate child rows if present.
            If Not oRow.ChildRows Is Nothing Then
                Call QueryBOMRowProperties(oRow.ChildRows, ItemTab)
            End If
        End If
    Next
    ItemTab = ItemTab - 3
End Sub
```

Exporting the assembly BOM

Description

This sample demonstrates exporting the Assembly BOM to an external file.

Code Samples

- VBA
- iLogic

This sample exports the structured and parts only views of the active assembly document to an Excel file. Other file formats are also supported.

Copy Code

```
Public Sub BOMExport()
    ' Set a reference to the assembly document.
    ' This assumes an assembly document is active.
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the BOM
    Dim oBOM As BOM
```

```
    Set oBOM = oDoc.ComponentDefinition.BOM

    ' Set the structured view to 'all levels'
    oBOM.StructuredViewFirstLevelOnly = False

    ' Make sure that the structured view is enabled.
    oBOM.StructuredViewEnabled = True

    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    oOptions.Add "TableName", "My BOMView"
    oOptions.Add "StartingCell", "A1"
    ' You can specify a template for the export.
    'oOptions.Add "Template", "C:\Temp\Template.xlsx"
    oOptions.Add "ExportedColumns", "Item;QTY;REV;Description;Part Number"
    oOptions.Add "AutoFitColumnWidth", True
    oOptions.Add "ApplyCellFormatting", True
    oOptions.Add "ForceCellToText", True

    ' Set a reference to the "Structured" BOMView
    Dim oStructuredBOMView As BOMView
    Set oStructuredBOMView = oBOM.BOMViews.Item("Structured")

    ' Export the BOM view to an Excel file
    oStructuredBOMView.Export "C:\temp\BOM-StructuredAllLevels.xls", kMicrosoftExcelFormat, oOptions

    ' Make sure that the parts only view is enabled.
    oBOM.PartsOnlyViewEnabled = True

    ' Set a reference to the "Parts Only" BOMView
    Dim oPartsOnlyBOMView As BOMView
    Set oPartsOnlyBOMView = oBOM.BOMViews.Item("Parts Only")

    ' Export the BOM view to an Excel file
    oPartsOnlyBOMView.Export "C:\temp\BOM-PartsOnly.xls", kMicrosoftExcelFormat, oOptions
End Sub
```

This sample exports the structured and parts only views of the active assembly document to an Excel file. Other file formats are also supported.

Copy Code

```
    ' Set a reference to the assembly document.
    ' This assumes an assembly document is active.
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the BOM
    Dim oBOM As BOM
    oBOM = oDoc.ComponentDefinition.BOM

    ' Set the structured view to 'all levels'
    oBOM.StructuredViewFirstLevelOnly = False

    ' Make sure that the structured view is enabled.
    oBOM.StructuredViewEnabled = True

    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    oOptions.Add("TableName", "My BOMView")
    oOptions.Add("StartingCell", "A1")
    ' You can specify a template for the export.
    'oOptions.Add("Template", "C:\Temp\Template.xlsx")
    oOptions.Add("ExportedColumns", "Item;QTY;REV;Description;Part Number")
    oOptions.Add("AutoFitColumnWidth", True)
    oOptions.Add("ApplyCellFormatting", True)
    oOptions.Add("ForceCellToText", True)

    ' Set a reference to the "Structured" BOMView
    Dim oStructuredBOMView As BOMView
    oStructuredBOMView = oBOM.BOMViews.Item("Structured")

    ' Export the BOM view to an Excel file
    oStructuredBOMView.Export("C:\temp\BOM-StructuredAllLevels.xls", kMicrosoftExcelFormat, oOptions)

    ' Make sure that the parts only view is enabled.
    oBOM.PartsOnlyViewEnabled = True

    ' Set a reference to the "Parts Only" BOMView
    Dim oPartsOnlyBOMView As BOMView
    oPartsOnlyBOMView = oBOM.BOMViews.Item("Parts Only")

    ' Export the BOM view to an Excel file
    oPartsOnlyBOMView.Export("C:\temp\BOM-PartsOnly.xls", kMicrosoftExcelFormat, oOptions)
```

Add assembly insert constraint

Description

This sample demonstrates the creation of an assembly insert constraint.

Code Samples

- VBA
- iLogic

Before running the sample, you need to open an assembly that contains at least two parts. Select circular edges on the two parts that will be used for the constraint and run the sample code. (Set the priority of the Select command and use the Shift-Select to select multiple edges.)

Copy Code

```
Public Sub InsertConstraint()
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    Set oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox "You must select the two circular edges for the insert."
        Exit Sub
    End If

    If Not TypeOf oSelectSet.Item(1) Is Edge Or Not TypeOf oSelectSet.Item(2) Is Edge Then
        MsgBox "You must select the two circular edges for the insert."
        Exit Sub
    End If

    ' Get the two edges from the select set.
    Dim oEdge1 As Edge
    Dim oEdge2 As Edge
    Set oEdge1 = oSelectSet.Item(1)
    Set oEdge2 = oSelectSet.Item(2)

    ' Create the insert constraint between the parts.
    Dim oInsert As InsertConstraint
    Set oInsert = oAsmCompDef.Constraints.AddInsertConstraint(oEdge1, oEdge2, True, 0)
End Sub
```

Before running the sample, you need to open an assembly that contains at least two parts. Select circular edges on the two parts that will be used for the constraint and run the sample code. (Set the priority of the Select command and use the Shift-Select to select multiple edges.)

Copy Code

```
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox("You must select the two circular edges for the insert.")
        Exit Sub
    End If

    If Not TypeOf oSelectSet.Item(1) Is Edge Or Not TypeOf oSelectSet.Item(2) Is Edge Then
        MsgBox("You must select the two circular edges for the insert.")
        Exit Sub
    End If

    ' Get the two edges from the select set.
    Dim oEdge1 As Edge
    Dim oEdge2 As Edge
    oEdge1 = oSelectSet.Item(1)
    oEdge2 = oSelectSet.Item(2)

    ' Create the insert constraint between the parts.
    Dim oInsert As InsertConstraint
    oInsert = oAsmCompDef.Constraints.AddInsertConstraint(oEdge1, oEdge2, True, 0)
```

Add assembly mate constraint

Description

This sample demonstrates the creation of an assembly mate constraint.

Code Samples

- VBA
- iLogic

Before running the sample, you need to open an assembly that contains at least two parts. Select planar faces on the two parts that will be used for the constraint and run the sample code. (Set the priority of the Select command and use the Shift-Select to select multiple faces.)

Copy Code

```
Public Sub MateConstraint()
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    Set oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox "You must select the two entities valid for mate."
        Exit Sub
    End If
```

```
    ' Get the two entities from the select set.
    Dim oBrepEnt1 As Object
    Dim oBrepEnt2 As Object
    Set oBrepEnt1 = oSelectSet.Item(1)
    Set oBrepEnt2 = oSelectSet.Item(2)

    ' Create the insert constraint between the parts.
    Dim oMate As MateConstraint
    Set oMate = oAsmCompDef.Constraints.AddMateConstraint(oBrepEnt1, oBrepEnt2, 0)
End Sub
```

Before running the sample, you need to open an assembly that contains at least two parts. Select planar faces on the two parts that will be used for the constraint and run the sample code. (Set the priority of the Select command and use the Shift-Select to select multiple faces.)

Copy Code

```
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox("You must select the two entities valid for mate.")
        Exit Sub
    End If

    ' Get the two entities from the select set.
    Dim oBrepEnt1 As Object
    Dim oBrepEnt2 As Object
    oBrepEnt1 = oSelectSet.Item(1)
    oBrepEnt2 = oSelectSet.Item(2)

    ' Create the insert constraint between the parts.
    Dim oMate As MateConstraint
    oMate = oAsmCompDef.Constraints.AddMateConstraint(oBrepEnt1, oBrepEnt2, 0)
```

Add mate constraint using work planes in parts

Description

This sample demonstrates creating a mate constraint between two occurrences using the work planes within those occurrences.

Code Samples

- VBA
- iLogic

To use the sample, have an assembly open that contains at least two occurrences, (part or subassembly), and run the program.

Copy Code

```
Public Sub MateConstraintOfWorkPlanes()
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get references to the two occurrences to constrain.
    ' This arbitrarily gets the first and second occurrence.
    Dim oOcc1 As ComponentOccurrence
    Set oOcc1 = oAsmCompDef.Occurrences.Item(1)

    Dim oOcc2 As ComponentOccurrence
    Set oOcc2 = oAsmCompDef.Occurrences.Item(2)

    ' Get the XY plane from each occurrence.  This goes to the
    ' component definition of the part to get this information.
    ' This is the same as accessing the part document directly.
    ' The work plane obtained is in the context of the part,
    ' not the assembly.
    Dim oPartPlane1 As WorkPlane
    Set oPartPlane1 = oOcc1.Definition.WorkPlanes.Item(3)

    Dim oPartPlane2 As WorkPlane
    Set oPartPlane2 = oOcc2.Definition.WorkPlanes.Item(3)

    ' Because we need the work plane in the context of the assembly
    ' we need to create proxies for the work planes.  The proxies
    ' represent the work planes in the context of the assembly.
    Dim oAsmPlane1 As WorkPlaneProxy
    Call oOcc1.CreateGeometryProxy(oPartPlane1, oAsmPlane1)

    Dim oAsmPlane2 As WorkPlaneProxy
    Call oOcc2.CreateGeometryProxy(oPartPlane2, oAsmPlane2)

    ' Create the constraint using the work plane proxies.
    Call oAsmCompDef.Constraints.AddMateConstraint(oAsmPlane1, oAsmPlane2, 0)
End Sub
```

To use the sample, have an assembly open that contains at least two occurrences, (part or subassembly), and run the program.

Copy Code

```
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition
```

```
    ' Get references to the two occurrences to constrain.
    ' This arbitrarily gets the first and second occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Item(1)

    Dim oOcc2 As ComponentOccurrence
    oOcc2 = oAsmCompDef.Occurrences.Item(2)

    ' Get the XY plane from each occurrence.  This goes to the
    ' component definition of the part to get this information.
    ' This is the same as accessing the part document directly.
    ' The work plane obtained is in the context of the part,
    ' not the assembly.
    Dim oPartPlane1 As WorkPlane
    oPartPlane1 = oOcc1.Definition.WorkPlanes.Item(3)

    Dim oPartPlane2 As WorkPlane
    oPartPlane2 = oOcc2.Definition.WorkPlanes.Item(3)

    ' Because we need the work plane in the context of the assembly
    ' we need to create proxies for the work planes.  The proxies
    ' represent the work planes in the context of the assembly.
    Dim oAsmPlane1 As WorkPlaneProxy
    Call oOcc1.CreateGeometryProxy(oPartPlane1, oAsmPlane1)

    Dim oAsmPlane2 As WorkPlaneProxy
    Call oOcc2.CreateGeometryProxy(oPartPlane2, oAsmPlane2)

    ' Create the constraint using the work plane proxies.
    Call oAsmCompDef.Constraints.AddMateConstraint(oAsmPlane1, oAsmPlane2, 0)
```

Add mate constraint with limits

Description

This sample demonstrates the creation of an assembly mate constraint with maximum and minimum limits defined.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub MateConstraintWithLimits()
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    Set oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox "You must select the two entities valid for mate."
        Exit Sub
    End If

    ' Get the two entities from the select set.
    Dim oBrepEnt1 As Object
    Dim oBrepEnt2 As Object
    Set oBrepEnt1 = oSelectSet.Item(1)
    Set oBrepEnt2 = oSelectSet.Item(2)

    ' Create the mate constraint between the parts, with an offset value of 0.
    Dim oMate As MateConstraint
    Set oMate = oAsmCompDef.Constraints.AddMateConstraint(oBrepEnt1, oBrepEnt2, 0)

    ' Set a maximum value of 2 inches
    oMate.ConstraintLimits.MaximumEnabled = True
    oMate.ConstraintLimits.Maximum.Expression = "2 in"

    ' Set a minimum value of -2 inches
    oMate.ConstraintLimits.MinimumEnabled = True
    oMate.ConstraintLimits.Minimum.Expression = "-2 in"
End Sub
```

Copy Code

```
    ' Set a reference to the assembly component definintion.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the select set.
    Dim oSelectSet As SelectSet
    oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' Validate the correct data is in the select set.
    If oSelectSet.Count <> 2 Then
        MsgBox("You must select the two entities valid for mate.")
        Exit Sub
    End If

    ' Get the two entities from the select set.
    Dim oBrepEnt1 As Object
    Dim oBrepEnt2 As Object
```

```
    oBrepEnt1 = oSelectSet.Item(1)
    oBrepEnt2 = oSelectSet.Item(2)

    ' Create the mate constraint between the parts, with an offset value of 0.
    Dim oMate As MateConstraint
    oMate = oAsmCompDef.Constraints.AddMateConstraint(oBrepEnt1, oBrepEnt2, 0)

    ' Set a maximum value of 2 inches
    oMate.ConstraintLimits.MaximumEnabled = True
    oMate.ConstraintLimits.Maximum.Expression = "2 in"

    ' Set a minimum value of -2 inches
    oMate.ConstraintLimits.MinimumEnabled = True
    oMate.ConstraintLimits.Minimum.Expression = "-2 in"
```

Create planar AssemblyJoint with offset to origins

Description

This sample demonstrates how to create a planar AssemblyJoint with offset to the OriginOne and OriginTwo.

Code Samples

- VBA
- iLogic

Create a part with some solid and make sure there are linear edges in it, save it as C:\Temp\Part1.ipt or you need to edit the VBA code to change the paths to make it work.

Copy Code

```
Sub CreateAssemblyJointWithOffsetSample()
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim oCompDef As AssemblyComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oMatrix As Matrix
    Set oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    Dim oOccu1 As ComponentOccurrence
    Dim oOccu2 As ComponentOccurrence
    ' Create two occurrences for adding assembly joint, make sure the sample Part1 has linear edge in it.
    Set oOccu1 = oCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)
    oMatrix.SetTranslation ThisApplication.TransientGeometry.CreateVector(20, 20, 20)
    Set oOccu2 = oCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)

    ' Create two GeometryIntent objects for creating assembly joint.
    Dim oOriginOne As GeometryIntent, oOriginTwo As GeometryIntent
    Dim oEdge As Edge
    For Each oEdge In oOccu1.SurfaceBodies(1).Edges
        If oEdge.GeometryType = kLineSegmentCurve Then
            Set oOriginOne = oCompDef.CreateGeometryIntent(oEdge, kMidPointIntent)
            Exit For
        End If
    Next

    For Each oEdge In oOccu2.SurfaceBodies(1).Edges
        If oEdge.GeometryType = kLineSegmentCurve Then
            Set oOriginTwo = oCompDef.CreateGeometryIntent(oEdge, kMidPointIntent)
            Exit For
        End If
    Next

    ' Create AssemblyJointDefinition
    Dim oJointDef As AssemblyJointDefinition
    Set oJointDef = oCompDef.Joints.CreateAssemblyJointDefinition(kPlanarJointType, oOriginOne, oOriginTwo)

    Call oJointDef.SetOriginOneAsOffset(5, 5)
    Call oJointDef.SetOriginTwoAsOffset(2, 2)

    Debug.Print oJointDef.OriginOneDefinitionType = kOffsetOriginDefinitionType

    ' Create assembly joint.
    Dim oJoint As AssemblyJoint
    Set oJoint = oCompDef.Joints.Add(oJointDef)
End Sub
```

Create a part with some solid and make sure there are linear edges in it, save it as C:\Temp\Part1.ipt or you need to edit the VBA code to change the paths to make it work.

Copy Code

```
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim oCompDef As AssemblyComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    Dim oOccu1 As ComponentOccurrence
    Dim oOccu2 As ComponentOccurrence
    ' Create two occurrences for adding assembly joint, make sure the sample Part1 has linear edge in it.
    oOccu1 = oCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)
    oMatrix.SetTranslation(ThisApplication.TransientGeometry.CreateVector(20, 20, 20))
```

```
    oOccu2 = oCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)

    ' Create two GeometryIntent objects for creating assembly joint.
    Dim oOriginOne As GeometryIntent, oOriginTwo As GeometryIntent
    Dim oEdge As Edge
    For Each oEdge In oOccu1.SurfaceBodies(1).Edges
        If oEdge.GeometryType = kLineSegmentCurve Then
            oOriginOne = oCompDef.CreateGeometryIntent(oEdge, kMidPointIntent)
            Exit For
        End If
    Next

    For Each oEdge In oOccu2.SurfaceBodies(1).Edges
        If oEdge.GeometryType = kLineSegmentCurve Then
            oOriginTwo = oCompDef.CreateGeometryIntent(oEdge, kMidPointIntent)
            Exit For
        End If
    Next

    ' Create AssemblyJointDefinition
    Dim oJointDef As AssemblyJointDefinition
    oJointDef = oCompDef.Joints.CreateAssemblyJointDefinition(kPlanarJointType, oOriginOne, oOriginTwo)

    Call oJointDef.SetOriginOneAsOffset(5, 5)
    Call oJointDef.SetOriginTwoAsOffset(2, 2)

    Logger.Info(oJointDef.OriginOneDefinitionType = kOffsetOriginDefinitionType)


    ' Create assembly joint.
    Dim oJoint As AssemblyJoint
    oJoint = oCompDef.Joints.Add(oJointDef)
```

Create rotational assembly joint

Description

This sample demonstrates creating an assembly joint. It connects the midpoints of the edges of two faces using a rotational joint. To do this it first creates a geometry intent object of the midpoint of the edge and then creates another intent using the face and the midpoint intent. It does this to create to midpoint intents which it then uses to create the rotational connection.

The sample uses and existing part that must be set up to allow it to work correctly. To create the sample part you can use any part that has a planar face and a linear edge connected to that planar face. A simple box is sufficient. In this part Add a mate iMate to the planar face and rename the iMate to "Face1". Also add a mate iMate to a linear edge that is on the face previously named and rename this iMate to "Edge1". Save the part to "C:\Temp\SamplePart.ipt" or any other name and edit the code below to reference the file. You can then run the sample code which will create a new assembly, insert two instances of the part and create a rotational connection between them. Then it will animation the rotation by driving the connection.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub AssemblyJointSample()
    ' Create a new assembly document.
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                 ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))
    Dim asmDef As AssemblyComponentDefinition
    Set asmDef = asmDoc.ComponentDefinition

    ' Place an occurrence into the assembly.
    Dim occ1 As ComponentOccurrence
    Dim occ2 As ComponentOccurrence
    Dim trans As Matrix
    Set trans = ThisApplication.TransientGeometry.CreateMatrix
    Set occ1 = asmDef.Occurrences.Add("C:\Temp\SamplePart.ipt", trans)

    ' Place a second occurrence with the matrix adjusted so it fits correctly with the first occurrence.
    trans.Cell(1, 4) = 6 * 2.54
    Set occ2 = asmDef.Occurrences.Add("C:\Temp\SamplePart.ipt", trans)

    ' Get Face 1 from occ1 and create a FaceProxy.
    Dim faceOnOcc1 As Face
    Set faceOnOcc1 = GetNamedEntity(occ1, "Face1")

    ' Get Face 1 from occ2 and create a FaceProxy.
    Dim faceOnOcc2 As Face
    Set faceOnOcc2 = GetNamedEntity(occ2, "Face1")

    ' Get Edge 1 from occ2 and create an EdgeProxy.
    Dim edgeOnOcc2 As Edge
    Set edgeOnOcc2 = GetNamedEntity(occ2, "Edge1")

    ' Get Edge 3 from occ1 and create an EdgeProxy.
    Dim edgeOnOcc1 As Edge
    Set edgeOnOcc1 = GetNamedEntity(occ1, "Edge1")

    ' Create an intent to the center of Edge1.
    Dim edgeOcc2Intent As GeometryIntent
    Set edgeOcc2Intent = asmDef.CreateGeometryIntent(edgeOnOcc2, PointIntentEnum.kMidPointIntent)

    ' Create an intent to the center of Edge3.
    Dim edgeOcc1Intent As GeometryIntent
    Set edgeOcc1Intent = asmDef.CreateGeometryIntent(edgeOnOcc1, PointIntentEnum.kMidPointIntent)
```

```
    ' Create two intents to define the geometry for the joint.
    Dim intentOne As GeometryIntent
    Set intentOne = asmDef.CreateGeometryIntent(faceOnOcc2, edgeOcc2Intent)
    Dim intentTwo As GeometryIntent
    Set intentTwo = asmDef.CreateGeometryIntent(faceOnOcc1, edgeOcc1Intent)

    ' Create a rotational jont between the two parts.
    Dim jointDef As AssemblyJointDefinition
    Set jointDef = asmDef.Joints.CreateAssemblyJointDefinition(kRotationalJointType, _
                                                               intentOne, intentTwo)
    jointDef.FlipAlignmentDirection = False
    jointDef.FlipOriginDirection = True
    Dim joint As AssemblyJoint
    Set joint = asmDef.Joints.Add(jointDef)

    ' Make the joint visible.
    joint.Visible = True

    ' Drive the joint to animate it.
    joint.DriveSettings.StartValue = "0 deg"
    joint.DriveSettings.EndValue = "180 deg"
    joint.DriveSettings.GoToStart
    joint.DriveSettings.PlayForward
    joint.DriveSettings.PlayReverse
End Sub


' This finds the entity associated with an iMate of a specified name.  This
' allows iMates to be used as a generic naming mechansim.
Private Function GetNamedEntity(Occurrence As ComponentOccurrence, Name As String) As Object
    ' Look for the iMate that has the specified name in the referenced file.
    Dim iMate As iMateDefinition
    Dim partDef As PartComponentDefinition
    Set partDef = Occurrence.Definition
    For Each iMate In partDef.iMateDefinitions
        ' Check to see if this iMate has the correct name.
        If UCase(iMate.Name) = UCase(Name) Then
            ' Get the geometry assocated with the iMate.
            Dim entity As Object
            Set entity = iMate.entity

            ' Create a proxy.
            Dim resultEntity As Object
            Set resultEntity = Nothing
            Call Occurrence.CreateGeometryProxy(entity, resultEntity)

            Exit For
        End If
    Next

    ' Return the found entity, or Nothing if a match wasn't found.
    Set GetNamedEntity = resultEntity
End Function
```

Copy Code

```
Sub Main
    ' Create a new assembly document.
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                 ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))
    Dim asmDef As AssemblyComponentDefinition
    asmDef = asmDoc.ComponentDefinition

    ' Place an occurrence into the assembly.
    Dim occ1 As ComponentOccurrence
    Dim occ2 As ComponentOccurrence
    Dim trans As Matrix
    trans = ThisApplication.TransientGeometry.CreateMatrix
    occ1 = asmDef.Occurrences.Add("C:\Temp\SamplePart.ipt", trans)

    ' Place a second occurrence with the matrix adjusted so it fits correctly with the first occurrence.
    trans.Cell(1, 4) = 6 * 2.54
    occ2 = asmDef.Occurrences.Add("C:\Temp\SamplePart.ipt", trans)

    ' Get Face 1 from occ1 and create a FaceProxy.
    Dim faceOnOcc1 As Face
    faceOnOcc1 = GetNamedEntity(occ1, "Face1")

    ' Get Face 1 from occ2 and create a FaceProxy.
    Dim faceOnOcc2 As Face
    faceOnOcc2 = GetNamedEntity(occ2, "Face1")

    ' Get Edge 1 from occ2 and create an EdgeProxy.
    Dim edgeOnOcc2 As Edge
    edgeOnOcc2 = GetNamedEntity(occ2, "Edge1")

    ' Get Edge 3 from occ1 and create an EdgeProxy.
    Dim edgeOnOcc1 As Edge
    edgeOnOcc1 = GetNamedEntity(occ1, "Edge1")

    ' Create an intent to the center of Edge1.
    Dim edgeOcc2Intent As GeometryIntent
    edgeOcc2Intent = asmDef.CreateGeometryIntent(edgeOnOcc2, PointIntentEnum.kMidPointIntent)

    ' Create an intent to the center of Edge3.
    Dim edgeOcc1Intent As GeometryIntent
    edgeOcc1Intent = asmDef.CreateGeometryIntent(edgeOnOcc1, PointIntentEnum.kMidPointIntent)

    ' Create two intents to define the geometry for the joint.
    Dim intentOne As GeometryIntent
    intentOne = asmDef.CreateGeometryIntent(faceOnOcc2, edgeOcc2Intent)
    Dim intentTwo As GeometryIntent
    intentTwo = asmDef.CreateGeometryIntent(faceOnOcc1, edgeOcc1Intent)

    ' Create a rotational jont between the two parts.
```

```
    Dim jointDef As AssemblyJointDefinition
    jointDef = asmDef.Joints.CreateAssemblyJointDefinition(kRotationalJointType, _
                                                           intentOne, intentTwo)
    jointDef.FlipAlignmentDirection = False
    jointDef.FlipOriginDirection = True
    Dim joint As AssemblyJoint
    joint = asmDef.Joints.Add(jointDef)

    ' Make the joint visible.
    joint.Visible = True

    ' Drive the joint to animate it.
    joint.DriveSettings.StartValue = "0 deg"
    joint.DriveSettings.EndValue = "180 deg"
    joint.DriveSettings.GoToStart
    joint.DriveSettings.PlayForward
    joint.DriveSettings.PlayReverse
End Sub


' This finds the entity associated with an iMate of a specified name.  This
' allows iMates to be used as a generic naming mechansim.
Private Function GetNamedEntity(Occurrence As ComponentOccurrence, Name As String) As Object
    ' Look for the iMate that has the specified name in the referenced file.
    Dim iMate As iMateDefinition
    Dim partDef As PartComponentDefinition
    partDef = Occurrence.Definition

        Dim resultEntity As Object
    For Each iMate In partDef.iMateDefinitions
        ' Check to see if this iMate has the correct name.
        If UCase(iMate.Name) = UCase(Name) Then
            ' Get the geometry assocated with the iMate.
            Dim entity As Object
            entity = iMate.entity

            ' Create a proxy.
            resultEntity = Nothing
            Call Occurrence.CreateGeometryProxy(entity, resultEntity)

            Exit For
        End If
    Next

    ' Return the found entity, or Nothing if a match wasn't found.
    GetNamedEntity = resultEntity
End Function
```

Add iMate Definition

Description

Add iMate definitions using AddMateiMateDefinition and AddInsertiMateDefinition.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateiMateDefinitionSample()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    Set oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
```

```
    Set oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                  oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Create a sketch circle with the center at (2, 1.5).
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(2, 1.5), 0.5)

    ' Create a profile.
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create the second extrude (a hole).
    Dim oExtrude2 As ExtrudeFeature
    Set oExtrude2 = oCompDef.Features.ExtrudeFeatures.AddByThroughAllExtent( _
                        oProfile, kNegativeExtentDirection, kCutOperation)

    ' Create a mate iMateDefinition on a side face of the first extrude.
    Dim oMateiMateDefinition As MateiMateDefinition
    Set oMateiMateDefinition = oCompDef.iMateDefinitions.AddMateiMateDefinition( _
                        oExtrude1.SideFaces.Item(1), 0, , , "MateA")

    ' Create a match list of names to use for the next iMateDefinition.
    Dim strMatchList(2) As String
    strMatchList(0) = "InsertA"
    strMatchList(1) = "InsertB"
    strMatchList(2) = "InsertC"

    ' Create an insert iMateDefinition on the cylindrical face of the second extrude.
    Dim oInsertiMateDefinition As InsertiMateDefinition
    Set oInsertiMateDefinition = oCompDef.iMateDefinitions.AddInsertiMateDefinition( _
                        oExtrude2.SideFaces.Item(1), False, 0, , "InsertA", strMatchList)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
               ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                  oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Create a sketch circle with the center at (2, 1.5).
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(2, 1.5), 0.5)

    ' Create a profile.
    oProfile = oSketch.Profiles.AddForSolid

    ' Create the second extrude (a hole).
    Dim oExtrude2 As ExtrudeFeature
    oExtrude2 = oCompDef.Features.ExtrudeFeatures.AddByThroughAllExtent( _
                        oProfile, kNegativeExtentDirection, kCutOperation)

    ' Create a mate iMateDefinition on a side face of the first extrude.
    Dim oMateiMateDefinition As MateiMateDefinition
    oMateiMateDefinition = oCompDef.iMateDefinitions.AddMateiMateDefinition( _
                        oExtrude1.SideFaces.Item(1), 0, , , "MateA")

    ' Create a match list of names to use for the next iMateDefinition.
    Dim strMatchList(2) As String
    strMatchList(0) = "InsertA"
    strMatchList(1) = "InsertB"
    strMatchList(2) = "InsertC"

    ' Create an insert iMateDefinition on the cylindrical face of the second extrude.
    Dim oInsertiMateDefinition As InsertiMateDefinition
```

```
    oInsertiMateDefinition = oCompDef.iMateDefinitions.AddInsertiMateDefinition( _
                        oExtrude2.SideFaces.Item(1), False, 0, , "InsertA", strMatchList)
```

iMate Result Creation

Description

This sample demonstrates creating an iMate result using two existin iMate definitions.

Code Samples

- VBA
- iLogic

To use this sample create a new part by extruding a rectangle to create a cube. Create a mate iMate on one of the faces. This sample assumes the iMate is named the default name used in the English version of Inventor, which is iMate:1. If the iMate definition is created with another name you can either edit the name of the iMate definition in the part file, or edit the sample code below to use the different name. Save the part to C:\Temp\iMatePart.ipt. Finally, have an assembly open and run the sample code.

Copy Code

```
Public Sub iMateResultCreationSample()
    ' Get the component definition of the currently open assembly.
    ' This will fail if an assembly document is not open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Create a new matrix object.  It will be initialized to an identity matrix.
    Dim oMatrix As Matrix
    Set oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    ' Place the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    Set oOcc1 = oAsmCompDef.Occurrences.Add("C:\Temp\iMatePart.ipt", oMatrix)

    ' Place the second occurrence, but adjust the matrix slightly so they're
    ' not right on top of each other.
    oMatrix.Cell(1, 4) = 10
    Dim oOcc2 As ComponentOccurrence
    Set oOcc2 = oAsmCompDef.Occurrences.Add("C:\Temp\iMatePart.ipt", oMatrix)

    ' Look through the iMateDefinitions defined for the first occurrence
    ' and find the one named "iMate:1".  This loop demonstrates using the
    ' Count and Item properties of the iMateDefinitions object.
    Dim i As Long
    Dim oiMateDef1 As iMateDefinition
    For i = 1 To oOcc1.iMateDefinitions.Count
        If oOcc1.iMateDefinitions.Item(i).Name = "iMate:1" Then
            Set oiMateDef1 = oOcc1.iMateDefinitions.Item(i)
            Exit For
        End If
    Next

    If oiMateDef1 Is Nothing Then
        MsgBox "An iMate definition named ""iMate:1"" does not exist in " & oOcc1.Name
        Exit Sub
    End If

    ' Look through the iMateDefinitions defined for the second occurrence
    ' and find the one named "iMate:1".  This loop demonstrates using the
    ' For Each method of iterating through a collection.
    Dim oiMateDef2 As iMateDefinition
    Dim bFoundDefinition As Boolean
    For Each oiMateDef2 In oOcc2.iMateDefinitions
        If oiMateDef2.Name = "iMate:1" Then
            bFoundDefinition = True
            Exit For
        End If
    Next

    If Not bFoundDefinition Then
        MsgBox "An iMate definition named ""iMate:1"" does not exist in " & oOcc2.Name
        Exit Sub
    End If

    ' Create an iMate result using the two definitions.
    Dim oiMateResult As iMateResult
    Set oiMateResult = oAsmCompDef.iMateResults.AddByTwoiMates(oiMateDef1, oiMateDef2)

End Sub
```

To use this sample create a new part by extruding a rectangle to create a cube. Create a mate iMate on one of the faces. This sample assumes the iMate is named the default name used in the English version of Inventor, which is iMate:1. If the iMate definition is created with another name you can either edit the name of the iMate definition in the part file, or edit the sample code below to use the different name. Save the part to C:\Temp\iMatePart.ipt. Finally, have an assembly open and run the sample code.

Copy Code

```
    ' Get the component definition of the currently open assembly.
    ' This will fail if an assembly document is not open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Create a new matrix object.  It will be initialized to an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix
```

```
    ' Place the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Add("C:\Temp\iMatePart.ipt", oMatrix)

    ' Place the second occurrence, but adjust the matrix slightly so they're
    ' not right on top of each other.
    oMatrix.Cell(1, 4) = 10
    Dim oOcc2 As ComponentOccurrence
    oOcc2 = oAsmCompDef.Occurrences.Add("C:\Temp\iMatePart.ipt", oMatrix)

    ' Look through the iMateDefinitions defined for the first occurrence
    ' and find the one named "iMate:1".  This loop demonstrates using the
    ' Count and Item properties of the iMateDefinitions object.
    Dim i As Long
    Dim oiMateDef1 As iMateDefinition
    For i = 1 To oOcc1.iMateDefinitions.Count
        If oOcc1.iMateDefinitions.Item(i).Name = "iMate:1" Then
            oiMateDef1 = oOcc1.iMateDefinitions.Item(i)
            Exit For
        End If
    Next

    If oiMateDef1 Is Nothing Then
        MsgBox("An iMate definition named ""iMate:1"" does not exist in " & oOcc1.Name)
        Exit Sub
    End If

    ' Look through the iMateDefinitions defined for the second occurrence
    ' and find the one named "iMate:1".  This loop demonstrates using the
    ' For Each method of iterating through a collection.
    Dim oiMateDef2 As iMateDefinition
    Dim bFoundDefinition As Boolean
    For Each oiMateDef2 In oOcc2.iMateDefinitions
        If oiMateDef2.Name = "iMate:1" Then
            bFoundDefinition = True
            Exit For
        End If
    Next

    If Not bFoundDefinition Then
        MsgBox("An iMate definition named ""iMate:1"" does not exist in " & oOcc2.Name)
        Exit Sub
    End If

    ' Create an iMate result using the two definitions.
    Dim oiMateResult As iMateResult
    oiMateResult = oAsmCompDef.iMateResults.AddByTwoiMates(oiMateDef1, oiMateDef2)
```

Add assembly occurrences to a new folder

Description

Demonstrates assembly occurrences to a new folder

Code Samples

- VBA
- iLogic

Have an assembly with at least one occurrence in it and run the sample.

Copy Code

```
Public Sub AddOccurrencesToFolder()
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    Dim oPane As BrowserPane
    Set oPane = oDoc.BrowserPanes.ActivePane

    Dim oOccurrenceNodes As ObjectCollection
    Set oOccurrenceNodes = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDef.Occurrences

        Dim oNode As BrowserNode
        Set oNode = oPane.GetBrowserNodeFromObject(oOcc)

        oOccurrenceNodes.Add oNode
    Next

    Dim oFolder As BrowserFolder
    Set oFolder = oPane.AddBrowserFolder("My Occurrence Folder", oOccurrenceNodes)
End Sub
```

Have an assembly with at least one occurrence in it and run the sample.

Copy Code

```
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    oDef = oDoc.ComponentDefinition
```

```
    Dim oPane As BrowserPane
    oPane = oDoc.BrowserPanes.ActivePane

    Dim oOccurrenceNodes As ObjectCollection
    oOccurrenceNodes = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDef.Occurrences

        Dim oNode As BrowserNode
        oNode = oPane.GetBrowserNodeFromObject(oOcc)

        oOccurrenceNodes.Add(oNode)
    Next

    Dim oFolder As BrowserFolder
    oFolder = oPane.AddBrowserFolder("My Occurrence Folder", oOccurrenceNodes)
```

Demote occurence

Description

This sample demonstrates how to demote a top level occurrence in an assembly into a new sub-assembly occurrence.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub Demote()
    ' Get the active assembly document
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    ' Get the occurrence to be demoted
    Dim oOcc As ComponentOccurrence
    Set oOcc = oDef.Occurrences.Item(1)

    ' Create a new sub-assembly to demote the occurrence into
    Dim oNewSubAssy As AssemblyDocument
    Set oNewSubAssy = ThisApplication.Documents.Add(kAssemblyDocumentObject, , False)

    Dim oMat As Matrix
    Set oMat = ThisApplication.TransientGeometry.CreateMatrix

    ' Create an instance of the new sub-assembly
    Dim oSubAssyOcc As ComponentOccurrence
    Set oSubAssyOcc = oDef.Occurrences.AddByComponentDefinition(oNewSubAssy.ComponentDefinition, oMat)

    ' Get the model browser
    Dim oPane As BrowserPane
    Set oPane = oDoc.BrowserPanes.Item("Model")

    ' Get the browser node that corresponds to the new sub-assembly occurrence
    Dim oSubAssyNode As BrowserNode
    Set oSubAssyNode = oPane.GetBrowserNodeFromObject(oSubAssyOcc)

    ' Get the last visible child node under the sub-assembly occurrence
    Dim oTargetNode As BrowserNode
    Dim i As Long
    For i = oSubAssyNode.BrowserNodes.Count To 1 Step -1
        If oSubAssyNode.BrowserNodes.Item(i).Visible Then
            Set oTargetNode = oSubAssyNode.BrowserNodes.Item(i)
            Exit For
        End If
    Next

    ' Get the browser node that corresponds to the occurrence to be demoted
    Dim oSourceNode As BrowserNode
    Set oSourceNode = oPane.GetBrowserNodeFromObject(oOcc)

    ' Demote the occurrence
    Call oPane.Reorder(oTargetNode, False, oSourceNode)
End Sub
```

Copy Code

```
    ' Get the active assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    oDef = oDoc.ComponentDefinition

    ' Get the occurrence to be demoted
    Dim oOcc As ComponentOccurrence
    oOcc = oDef.Occurrences.Item(1)

    ' Create a new sub-assembly to demote the occurrence into
    Dim oNewSubAssy As AssemblyDocument
    oNewSubAssy = ThisApplication.Documents.Add(kAssemblyDocumentObject, , False)
```

```
    Dim oMat As Matrix
    oMat = ThisApplication.TransientGeometry.CreateMatrix

    ' Create an instance of the new sub-assembly
    Dim oSubAssyOcc As ComponentOccurrence
    oSubAssyOcc = oDef.Occurrences.AddByComponentDefinition(oNewSubAssy.ComponentDefinition, oMat)

    ' Get the model browser
    Dim oPane As BrowserPane
    oPane = oDoc.BrowserPanes.Item("Model")

    ' Get the browser node that corresponds to the new sub-assembly occurrence
    Dim oSubAssyNode As BrowserNode
    oSubAssyNode = oPane.GetBrowserNodeFromObject(oSubAssyOcc)

    ' Get the last visible child node under the sub-assembly occurrence
    Dim oTargetNode As BrowserNode
    Dim i As Long
    For i = oSubAssyNode.BrowserNodes.Count To 1 Step -1
        If oSubAssyNode.BrowserNodes.Item(i).Visible Then
            oTargetNode = oSubAssyNode.BrowserNodes.Item(i)
            Exit For
        End If
    Next

    ' Get the browser node that corresponds to the occurrence to be demoted
    Dim oSourceNode As BrowserNode
    oSourceNode = oPane.GetBrowserNodeFromObject(oOcc)

    ' Demote the occurrence
    Call oPane.Reorder(oTargetNode, False, oSourceNode)
```

Promote occurence

Description

This sample demonstrates how to promote an occurrence.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub Promote()
    ' Get the active assembly document
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    ' Get the top level occurrence of an assembly
    Dim oSubAssyOcc As ComponentOccurrence
    Set oSubAssyOcc = oDef.Occurrences.Item(1)

    ' Get the 2nd level occurrence under the assembly occurrence
    Dim oSubOcc As ComponentOccurrenceProxy
    Set oSubOcc = oDef.Occurrences.Item(1).SubOccurrences.Item(1)

    Dim oPane As BrowserPane
    Set oPane = oDoc.BrowserPanes.Item("Model")

    ' Get the browser nodes corresponding to the two occurrences
    Dim oTargetNode As BrowserNode
    Set oTargetNode = oPane.GetBrowserNodeFromObject(oSubAssyOcc)

    Dim oSourceNode As BrowserNode
    Set oSourceNode = oPane.GetBrowserNodeFromObject(oSubOcc)

    ' Reorder the nodes to promote the sub-occurrence to the top level
    Call oPane.Reorder(oTargetNode, True, oSourceNode)
End Sub
```

Copy Code

```
    ' Get the active assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    oDef = oDoc.ComponentDefinition

    ' Get the top level occurrence of an assembly
    Dim oSubAssyOcc As ComponentOccurrence
    oSubAssyOcc = oDef.Occurrences.Item(1)

    ' Get the 2nd level occurrence under the assembly occurrence
    Dim oSubOcc As ComponentOccurrenceProxy
    oSubOcc = oDef.Occurrences.Item(1).SubOccurrences.Item(1)

    Dim oPane As BrowserPane
    oPane = oDoc.BrowserPanes.Item("Model")

    ' Get the browser nodes corresponding to the two occurrences
    Dim oTargetNode As BrowserNode
```

```
    oTargetNode = oPane.GetBrowserNodeFromObject(oSubAssyOcc)

    Dim oSourceNode As BrowserNode
    oSourceNode = oPane.GetBrowserNodeFromObject(oSubOcc)

    ' Reorder the nodes to promote the sub-occurrence to the top level
    Call oPane.Reorder(oTargetNode, True, oSourceNode)
```

Assembly Ground Occurrences

Description

This sample demonstrates grounding an assembly occurrence.

Code Samples

- VBA
- iLogic

Before running the sample, you need to open an assembly and create a part file called C:\TempPart1.ipt, or edit the sample code to point to another part file if desired.

Copy Code

```
Public Sub FixAllOccurrences()
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Ask whether to delete or suppress the existing constraints.
    Dim bDelete As Boolean
    If MsgBox("Do you want to delete all existing constraints?", vbYesNo + vbQuestion) = vbYes Then
      bDelete = True
    Else
      bDelete = False
    End If

    ' Iterate through all of the constraints and perform the specified operation.
    Dim oConstraint As AssemblyConstraint
    For Each oConstraint In oAsmCompDef.Constraints
      If bDelete Then
        oConstraint.Delete
      Else
        oConstraint.Suppressed = True
      End If
    Next

    ' Iterate through all of the occurrences and ground them.
    Dim oOccurrence As ComponentOccurrence
    For Each oOccurrence In oAsmCompDef.Occurrences
      oOccurrence.Grounded = True
    Next
End Sub
```

Before running the sample, you need to open an assembly and create a part file called C:\TempPart1.ipt, or edit the sample code to point to another part file if desired.

Copy Code

```
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Ask whether to delete or suppress the existing constraints.
    Dim bDelete As Boolean
    If MsgBox("Do you want to delete all existing constraints?", vbYesNo + vbQuestion) = vbYes Then
      bDelete = True
    Else
      bDelete = False
    End If

    ' Iterate through all of the constraints and perform the specified operation.
    Dim oConstraint As AssemblyConstraint
    For Each oConstraint In oAsmCompDef.Constraints
      If bDelete Then
        oConstraint.Delete
      Else
        oConstraint.Suppressed = True
      End If
    Next

    ' Iterate through all of the occurrences and ground them.
    Dim oOccurrence As ComponentOccurrence
    For Each oOccurrence In oAsmCompDef.Occurrences
      oOccurrence.Grounded = True
    Next
```

Create Revit Export sample

Description

This sample demonstrates how to create a RevitExport object.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a RevitExport object. Open an assembly firstly before running this sample.

Copy Code

```
Sub CreateRevitExportSample()
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Actiate the Master model state if the active model state is substitute.
    If oDoc.ComponentDefinition.ModelStates.ActiveModelState.ModelStateType = ModelStateTypeEnum.kSubstituteModelStateType Then
        oDoc.ComponentDefinition.ModelStates.Item(1).Activate
        Set oDoc = ThisApplication.ActiveDocument
    End If

    Dim oRevitExportDef As RevitExportDefinition
    Set oRevitExportDef = oDoc.ComponentDefinition.RevitExports.CreateDefinition

    oRevitExportDef.Location = "C:\Temp"
    oRevitExportDef.FileName = "MyRevitExport.rvt"
    oRevitExportDef.Structure = kEachTopLevelComponentStructure
    oRevitExportDef.EnableUpdating = True

    ' Create RevitExport.
    Dim oRevitExport As RevitExport
    Set oRevitExport = oDoc.ComponentDefinition.RevitExports.Add(oRevitExportDef)
End Sub
```

This sample demonstrates how to create a RevitExport object. Open an assembly firstly before running this sample.

Copy Code

```
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    ' Actiate the Master model state if the active model state is substitute.
    If oDoc.ComponentDefinition.ModelStates.ActiveModelState.ModelStateType = ModelStateTypeEnum.kSubstituteModelStateType Then
        oDoc.ComponentDefinition.ModelStates.Item(1).Activate
        oDoc = ThisApplication.ActiveDocument
    End If

    Dim oRevitExportDef As RevitExportDefinition
    oRevitExportDef = oDoc.ComponentDefinition.RevitExports.CreateDefinition

    oRevitExportDef.Location = "C:\Temp"
    oRevitExportDef.FileName = "MyRevitExport.rvt"
    oRevitExportDef.Structure = kEachTopLevelComponentStructure
    oRevitExportDef.EnableUpdating = True

    ' Create RevitExport.
    Dim oRevitExport As RevitExport
    oRevitExport = oDoc.ComponentDefinition.RevitExports.Add(oRevitExportDef)
```

Open assembly using last model state

Description

This sample demonstrates how to open an assembly document in its last active model state.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub OpenDocumentInLastActiveModelState()
    Dim strFullFileName As String
    strFullFileName = "C:\temp\Assembly1.iam"

    ' Set a reference to the FileManager object.
    Dim oFileManager As FileManager
    Set oFileManager = ThisApplication.FileManager

    ' Get the name of the last active model state.
    Dim strLastActiveMS As String
    strLastActiveMS = oFileManager.GetLastActiveModelState(strFullFileName)

    ' Use the full file name and ModelState name to get the full document name.
    Dim strFullDocumentName As String
    strFullDocumentName = oFileManager.GetFullDocumentName(strFullFileName, strLastActiveMS)

    ' Open the document in the last active model state.
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.Documents.Open(strFullDocumentName)
End Sub
```

Copy Code

```
    Dim strFullFileName As String
    strFullFileName = "C:\temp\Assembly1.iam"
```

```
    ' Set a reference to the FileManager object.
    Dim oFileManager As FileManager
    oFileManager = ThisApplication.FileManager

    ' Get the name of the last active model state.
    Dim strLastActiveMS As String
    strLastActiveMS = oFileManager.GetLastActiveModelState(strFullFileName)

    ' Use the full file name and ModelState name to get the full document name.
    Dim strFullDocumentName As String
    strFullDocumentName = oFileManager.GetFullDocumentName(strFullFileName, strLastActiveMS)

    ' Open the document in the last active model state.
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Open(strFullDocumentName)
```

Create a model state

Description

This sample demonstrates creation of a model state in an assembly.

Code Samples

- VBA
- iLogic

Before running the sample, you need to open an assembly and create two part files called C:\Temp\Part1.ipt and C:\Temp\Part2.ipt, or edit the sample code to point to different part files if desired.

Copy Code

```
Public Sub CreateModelState()
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create a matrix. A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    Set oMatrix = oTG.CreateMatrix

    ' Add the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    Set oOcc1 = oAsmCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)

    ' Set the translation portion of the matrix so the
    ' second part will be positioned at (3,2,1).
    Call oMatrix.SetTranslation(oTG.CreateVector(3, 2, 1))

    ' Add the second occurrence.
    Dim oOcc2 As ComponentOccurrence
    Set oOcc2 = oAsmCompDef.Occurrences.Add("C:\Temp\Part2.ipt", oMatrix)

    ' Create a new level of detail representation.
    ' The new representation is automatically activated.
    Dim oModelState As ModelState
    Set oModelState = oAsmCompDef.ModelStates.Add("Part2Suppressed")

    ' Suppress the second component in the new model state.
    ' If the document "C:\Temp\Part2.ipt" is not currently referenced
    ' elsewhere, it will be closed.
    oOcc2.Suppress
End Sub
```

Before running the sample, you need to open an assembly and create two part files called C:\Temp\Part1.ipt and C:\Temp\Part2.ipt, or edit the sample code to point to different part files if desired.

Copy Code

```
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create a matrix. A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = oTG.CreateMatrix

    ' Add the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)

    ' Set the translation portion of the matrix so the
    ' second part will be positioned at (3,2,1).
    Call oMatrix.SetTranslation(oTG.CreateVector(3, 2, 1))

    ' Add the second occurrence.
    Dim oOcc2 As ComponentOccurrence
```

```
    oOcc2 = oAsmCompDef.Occurrences.Add("C:\Temp\Part2.ipt", oMatrix)

    ' Create a new level of detail representation.
    ' The new representation is automatically activated.
    Dim oModelState As ModelState
    oModelState = oAsmCompDef.ModelStates.Add("Part2Suppressed")

    ' Suppress the second component in the new model state.
    ' If the document "C:\Temp\Part2.ipt" is not currently referenced
    ' elsewhere, it will be closed.
    oOcc2.Suppress
```

Associative body copy

Description

The following sample demonstrates copying bodies (associatively and non-associatively) across parts in an assembly.

Code Samples

- VBA
- iLogic

Before running the sample, create an assembly with two parts in it. The sample copies the body from the first part into the second part.

Copy Code

```
Sub AssociativeBodyCopy()
    ' Set a reference to the active assembly document.
    Dim oAssemblyDoc As AssemblyDocument
    Set oAssemblyDoc = ThisApplication.ActiveDocument

    Dim oAssemblyDef As AssemblyComponentDefinition
    Set oAssemblyDef = oAssemblyDoc.ComponentDefinition

    Dim oOccurrence1 As ComponentOccurrence
    Set oOccurrence1 = oAssemblyDef.Occurrences.Item(1)

    Dim oPartDef1 As PartComponentDefinition
    Set oPartDef1 = oOccurrence1.Definition

    Dim oOccurrence2 As ComponentOccurrence
    Set oOccurrence2 = oAssemblyDef.Occurrences.Item(2)

    Dim oPartDef2 As PartComponentDefinition
    Set oPartDef2 = oOccurrence2.Definition

    ' Get the source solid body from the first part.
    Dim oSourceBody As SurfaceBody
    Set oSourceBody = oPartDef1.SurfaceBodies.Item(1)

    Dim oSourceBodyProxy As SurfaceBodyProxy
    Call oOccurrence1.CreateGeometryProxy(oSourceBody, oSourceBodyProxy)

    ' Create an associative surface base feature in the second part.
    Dim oFeatureDef1 As NonParametricBaseFeatureDefinition
    Set oFeatureDef1 = oPartDef2.Features.NonParametricBaseFeatures.CreateDefinition

    Dim oCollection As ObjectCollection
    Set oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    oCollection.Add oSourceBodyProxy

    oFeatureDef1.BRepEntities = oCollection
    oFeatureDef1.OutputType = kSurfaceOutputType
    oFeatureDef1.TargetOccurrence = oOccurrence2
    oFeatureDef1.IsAssociative = True

    Dim oBaseFeature1 As NonParametricBaseFeature
    Set oBaseFeature1 = oPartDef2.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef1)

    ' Create a non-associative solid base feature in the second part.
    Dim oFeatureDef2 As NonParametricBaseFeatureDefinition
    Set oFeatureDef2 = oPartDef2.Features.NonParametricBaseFeatures.CreateDefinition

    oFeatureDef2.BRepEntities = oCollection
    oFeatureDef2.OutputType = kSolidOutputType
    oFeatureDef2.TargetOccurrence = oOccurrence2

    Dim oBaseFeature2 As NonParametricBaseFeature
    Set oBaseFeature2 = oPartDef2.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef2)

    oAssemblyDoc.Update
End Sub
```

Before running the sample, create an assembly with two parts in it. The sample copies the body from the first part into the second part.

Copy Code

```
    ' Set a reference to the active assembly document.
    Dim oAssemblyDoc As AssemblyDocument
    oAssemblyDoc = ThisApplication.ActiveDocument

    Dim oAssemblyDef As AssemblyComponentDefinition
    oAssemblyDef = oAssemblyDoc.ComponentDefinition

    Dim oOccurrence1 As ComponentOccurrence
    oOccurrence1 = oAssemblyDef.Occurrences.Item(1)
```

```
    Dim oPartDef1 As PartComponentDefinition
    oPartDef1 = oOccurrence1.Definition

    Dim oOccurrence2 As ComponentOccurrence
    oOccurrence2 = oAssemblyDef.Occurrences.Item(2)

    Dim oPartDef2 As PartComponentDefinition
    oPartDef2 = oOccurrence2.Definition

    ' Get the source solid body from the first part.
    Dim oSourceBody As SurfaceBody
    oSourceBody = oPartDef1.SurfaceBodies.Item(1)

    Dim oSourceBodyProxy As SurfaceBodyProxy
    Call oOccurrence1.CreateGeometryProxy(oSourceBody, oSourceBodyProxy)

    ' Create an associative surface base feature in the second part.
    Dim oFeatureDef1 As NonParametricBaseFeatureDefinition
    oFeatureDef1 = oPartDef2.Features.NonParametricBaseFeatures.CreateDefinition

    Dim oCollection As ObjectCollection
    oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    oCollection.Add(oSourceBodyProxy)

    oFeatureDef1.BRepEntities = oCollection
    oFeatureDef1.OutputType = kSurfaceOutputType
    oFeatureDef1.TargetOccurrence = oOccurrence2
    oFeatureDef1.IsAssociative = True

    Dim oBaseFeature1 As NonParametricBaseFeature
    oBaseFeature1 = oPartDef2.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef1)

    ' Create a non-associative solid base feature in the second part.
    Dim oFeatureDef2 As NonParametricBaseFeatureDefinition
    oFeatureDef2 = oPartDef2.Features.NonParametricBaseFeatures.CreateDefinition

    oFeatureDef2.BRepEntities = oCollection
    oFeatureDef2.OutputType = kSolidOutputType
    oFeatureDef2.TargetOccurrence = oOccurrence2

    Dim oBaseFeature2 As NonParametricBaseFeature
    oBaseFeature2 = oPartDef2.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef2)

    oAssemblyDoc.Update
```

Shrink wrap substitute in assembly

Description

The following sample demonstrates the creation of a shrinkwrap substitute within an assembly.

Code Samples

- VBA
- iLogic

Open any assembly document and run the sample. A shrinkwrap part is created at the same location as the assembly.

Copy Code

```
Sub CreateShrinkwrapSubstitute()
    ' Set a reference to the active assembly document
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    ' Create a new part document that will be the shrinkwrap substitute
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, , False)

    Dim oPartDef As PartComponentDefinition
    Set oPartDef = oPartDoc.ComponentDefinition

    Dim oDerivedAssemblyDef As DerivedAssemblyDefinition
    Set oDerivedAssemblyDef = oPartDef.ReferenceComponents.DerivedAssemblyComponents.CreateDefinition(oDoc.FullDocumentName)

    ' Set various shrinkwrap related options
    oDerivedAssemblyDef.DeriveStyle = kDeriveAsSingleBodyNoSeams
    oDerivedAssemblyDef.IncludeAllTopLevelWorkFeatures = kDerivedIncludeAll
    oDerivedAssemblyDef.IncludeAllTopLevelSketches = kDerivedIncludeAll
    oDerivedAssemblyDef.IncludeAllTopLeveliMateDefinitions = kDerivedExcludeAll
    oDerivedAssemblyDef.IncludeAllTopLevelParameters = kDerivedExcludeAll
    oDerivedAssemblyDef.ReducedMemoryMode = True

    Call oDerivedAssemblyDef.SetHolePatchingOptions(kDerivedPatchAll)
    Call oDerivedAssemblyDef.SetRemoveByVisibilityOptions(kDerivedRemovePartsAndFaces, 25)

    ' Create the shrinkwrap component
    Dim oDerivedAssembly As DerivedAssemblyComponent
    Set oDerivedAssembly = oPartDef.ReferenceComponents.DerivedAssemblyComponents.Add(oDerivedAssemblyDef)

    ' Save the part
    Dim strSubstituteFileName As String
    strSubstituteFileName = Left$(oDoc.FullFileName, Len(oDoc.FullFileName) - 4)
    strSubstituteFileName = strSubstituteFileName & "_ShrinkwrapSubstitute.ipt"
```

```
    ThisApplication.SilentOperation = True
    Call oPartDoc.SaveAs(strSubstituteFileName, False)
    ThisApplication.SilentOperation = False

    ' Create a substitute level of detail using the shrinkwrap part.
    Dim oSubstituteMS As ModelState
    Set oSubstituteMS = oDef.ModelStates.AddSubstitute(strSubstituteFileName)

    ' Release reference of the invisibly opened part document.
    oPartDoc.ReleaseReference
End Sub
```

Open any assembly document and run the sample. A shrinkwrap part is created at the same location as the assembly.

Copy Code

```
    ' Set a reference to the active assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oDef As AssemblyComponentDefinition
    oDef = oDoc.ComponentDefinition

    ' Create a new part document that will be the shrinkwrap substitute
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, , False)

    Dim oPartDef As PartComponentDefinition
    oPartDef = oPartDoc.ComponentDefinition

    Dim oDerivedAssemblyDef As DerivedAssemblyDefinition
    oDerivedAssemblyDef = oPartDef.ReferenceComponents.DerivedAssemblyComponents.CreateDefinition(oDoc.FullDocumentName)

    ' Set various shrinkwrap related options
    oDerivedAssemblyDef.DeriveStyle = kDeriveAsSingleBodyNoSeams
    oDerivedAssemblyDef.IncludeAllTopLevelWorkFeatures = kDerivedIncludeAll
    oDerivedAssemblyDef.IncludeAllTopLevelSketches = kDerivedIncludeAll
    oDerivedAssemblyDef.IncludeAllTopLeveliMateDefinitions = kDerivedExcludeAll
    oDerivedAssemblyDef.IncludeAllTopLevelParameters = kDerivedExcludeAll
    oDerivedAssemblyDef.ReducedMemoryMode = True

    Call oDerivedAssemblyDef.SetHolePatchingOptions(kDerivedPatchAll)
    Call oDerivedAssemblyDef.SetRemoveByVisibilityOptions(kDerivedRemovePartsAndFaces, 25)

    ' Create the shrinkwrap component
    Dim oDerivedAssembly As DerivedAssemblyComponent
    oDerivedAssembly = oPartDef.ReferenceComponents.DerivedAssemblyComponents.Add(oDerivedAssemblyDef)

    ' Save the part
    Dim strSubstituteFileName As String
    strSubstituteFileName = Left$(oDoc.FullFileName, Len(oDoc.FullFileName) - 4)
    strSubstituteFileName = strSubstituteFileName & "_ShrinkwrapSubstitute.ipt"

    ThisApplication.SilentOperation = True
    Call oPartDoc.SaveAs(strSubstituteFileName, False)
    ThisApplication.SilentOperation = False

    ' Create a substitute level of detail using the shrinkwrap part.
    Dim oSubstituteMS As ModelState
    oSubstituteMS = oDef.ModelStates.AddSubstitute(strSubstituteFileName)

    ' Release reference of the invisibly opened part document.
    oPartDoc.ReleaseReference
```

Adding iAssembly occurrences

Description

This sample demonstrates adding iAssembly occurrences to an assembly.

Code Samples

- VBA
- iLogic
- C#

Before running the sample, make sure that C:\temp\iAssemblyFactory.iam exists and that it is an iAssembly factory.

Copy Code

```
Public Sub AddiAssemblyOccurrence()
    ' Open the factory document invisible.
    Dim oFactoryDoc As AssemblyDocument
    Set oFactoryDoc = ThisApplication.Documents.Open("C:\temp\iAssemblyFactory.iam", False)

    ' Set a reference to the component definition.
    Dim oCompDef As AssemblyComponentDefinition
    Set oCompDef = oFactoryDoc.ComponentDefinition

    ' Make sure we have an iAssembly factory.
    If oCompDef.IsiAssemblyFactory = False Then
        MsgBox "Chosen document is not a factory.", vbExclamation
        Exit Sub
    End If

    ' Set a reference to the factory.
    Dim oiAssyFactory As iAssemblyFactory
    Set oiAssyFactory = oCompDef.iAssemblyFactory
```

```
    ' Get the number of rows in the factory.
    Dim iNumRows As Integer
    iNumRows = oiAssyFactory.TableRows.Count

    ' Create a new assembly document
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, , True)

    Dim oOccs As ComponentOccurrences
    Set oOccs = oDoc.ComponentDefinition.Occurrences

    Dim oPos As Matrix
    Set oPos = ThisApplication.TransientGeometry.CreateMatrix

    Dim oStep As Double
    oStep = 0#
    Dim iRow As Long

    ' Add an occurrence for each member in the factory.
    For iRow = 1 To iNumRows

        oStep = oStep + 10

        ' Add a translation along X axis
        oPos.SetTranslation ThisApplication.TransientGeometry.CreateVector(oStep, oStep, 0)

        Dim oOcc As ComponentOccurrence
        Set oOcc = oOccs.AddiAssemblyMember("C:\temp\iAssemblyFactory.iam ", oPos, iRow)
    Next
End Sub
```

Before running the sample, make sure that C:\temp\iAssemblyFactory.iam exists and that it is an iAssembly factory.

Copy Code

```
    ' Open the factory document invisible.
    Dim oFactoryDoc As AssemblyDocument
    oFactoryDoc = ThisApplication.Documents.Open("C:\temp\iAssemblyFactory.iam", False)

    ' Set a reference to the component definition.
    Dim oCompDef As AssemblyComponentDefinition
    oCompDef = oFactoryDoc.ComponentDefinition

    ' Make sure we have an iAssembly factory.
    If oCompDef.IsiAssemblyFactory = False Then
        MsgBox("Chosen document is not a factory.", vbExclamation)
        Exit Sub
    End If

    ' Set a reference to the factory.
    Dim oiAssyFactory As iAssemblyFactory
    oiAssyFactory = oCompDef.iAssemblyFactory

    ' Get the number of rows in the factory.
    Dim iNumRows As Integer
    iNumRows = oiAssyFactory.TableRows.Count

    ' Create a new assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, , True)

    Dim oOccs As ComponentOccurrences
    oOccs = oDoc.ComponentDefinition.Occurrences

    Dim oPos As Matrix
    oPos = ThisApplication.TransientGeometry.CreateMatrix

    Dim oStep As Double
    oStep = 0#
    Dim iRow As Long

    ' Add an occurrence for each member in the factory.
    For iRow = 1 To iNumRows

        oStep = oStep + 10

        ' Add a translation along X axis
        oPos.SetTranslation(ThisApplication.TransientGeometry.CreateVector(oStep, oStep, 0))

        Dim oOcc As ComponentOccurrence
        oOcc = oOccs.AddiAssemblyMember("C:\temp\iAssemblyFactory.iam ", oPos, iRow)
    Next
```

Before running the sample, make sure that C:/temp/iAssemblyFactory.iam exists and that it is an iAssembly factory. The first line of the C# sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void AddiAssemblyOccurrence()
{
    Application oApp = ThisApplication;

    // Open the factory document invisible.
    AssemblyDocument oFactoryDoc = (AssemblyDocument)oApp.Documents.Open("C:/temp/iAssemblyFactory.iam", false);

    // Set a reference to the component definition.
    AssemblyComponentDefinition oCompDef = oFactoryDoc.ComponentDefinition;

    // Make sure we have an iAssembly factory.
    if(!oCompDef.IsiAssemblyFactory)
    {
        System.Windows.Forms.MessageBox.Show("Chosen document is not a factory.", "Invalid document");
        return;
    }

    // Set a reference to the factory.
```

```
    iAssemblyFactory oiAssyFactory = oCompDef.iAssemblyFactory;

    // Get the number of rows in the factory.
    int iNumRows = oiAssyFactory.TableRows.Count;

    // Create a new assembly document
    AssemblyDocument oDoc = (AssemblyDocument)oApp.Documents.Add(DocumentTypeEnum.kAssemblyDocumentObject, "" , true);

    ComponentOccurrences oOccs = oDoc.ComponentDefinition.Occurrences;

    Matrix oPos = oApp.TransientGeometry.CreateMatrix();

    int oStep = 0;
    int iRow;

    // Add an occurrence for each member in the factory.
    for (iRow = 1; iRow <= iNumRows; iRow++)
    {
        oStep = oStep + 10;

        // Add a translation along X axis
        oPos.SetTranslation(oApp.TransientGeometry.CreateVector(oStep, oStep, 0), false);

        ComponentOccurrence oOcc = oOccs.AddiAssemblyMember("C:/temp/iAssemblyFactory.iam ", oPos, iRow, null);
    }

    // Release reference of the invisibly open document
    oFactoryDoc.ReleaseReference();

}
```

Adding iPart occurrences to an assembly

Description

This sample demonstrates adding iPart occurrences to an assembly.

Code Samples

- VBA
- iLogic
- C#

Before running the sample, make sure that C:\temp\iPartFactory.ipt exists and that it is an iPart factory.

Copy Code

```
Public Sub AddiPartOccurrence()
    ' Open the factory document invisible.
    Dim oFactoryDoc As PartDocument
    Set oFactoryDoc = ThisApplication.Documents.Open("C:\temp\iPartFactory.ipt", False)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oFactoryDoc.ComponentDefinition

    ' Make sure we have an iPart factory.
    If oCompDef.IsiPartFactory = False Then
        MsgBox "Chosen document is not a factory.", vbExclamation
        Exit Sub
    End If

    ' Set a reference to the factory.
    Dim oiPartFactory As iPartFactory
    Set oiPartFactory = oCompDef.iPartFactory

    ' Get the number of rows in the factory.
    Dim iNumRows As Integer
    iNumRows = oiPartFactory.TableRows.Count

    ' Create a new assembly document
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, , True)

    Dim oOccs As ComponentOccurrences
    Set oOccs = oDoc.ComponentDefinition.Occurrences

    Dim oPos As Matrix
    Set oPos = ThisApplication.TransientGeometry.CreateMatrix

    Dim oStep As Double
    oStep = 0#
    Dim iRow As Long

    ' Add an occurrence for each member in the factory.
    For iRow = 1 To iNumRows

        oStep = oStep + 10

        ' Add a translation along X axis
        oPos.SetTranslation ThisApplication.TransientGeometry.CreateVector(oStep, oStep, 0)

        Dim oOcc As ComponentOccurrence
        Set oOcc = oOccs.AddiPartMember("C:\temp\iPartFactory.ipt ", oPos, iRow)
    Next
End Sub
```

Before running the sample, make sure that C:\temp\iPartFactory.ipt exists and that it is an iPart factory.

Copy Code

```
    ' Open the factory document invisible.
    Dim oFactoryDoc As PartDocument
    oFactoryDoc = ThisApplication.Documents.Open("C:\temp\iPartFactory.ipt", False)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oFactoryDoc.ComponentDefinition

    ' Make sure we have an iPart factory.
    If oCompDef.IsiPartFactory = False Then
        MsgBox("Chosen document is not a factory.", vbExclamation)
        Exit Sub
    End If

    ' Set a reference to the factory.
    Dim oiPartFactory As iPartFactory
    oiPartFactory = oCompDef.iPartFactory

    ' Get the number of rows in the factory.
    Dim iNumRows As Integer
    iNumRows = oiPartFactory.TableRows.Count

    ' Create a new assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, , True)

    Dim oOccs As ComponentOccurrences
    oOccs = oDoc.ComponentDefinition.Occurrences

    Dim oPos As Matrix
    oPos = ThisApplication.TransientGeometry.CreateMatrix

    Dim oStep As Double
    oStep = 0#
    Dim iRow As Long

    ' Add an occurrence for each member in the factory.
    For iRow = 1 To iNumRows

        oStep = oStep + 10

        ' Add a translation along X axis
        oPos.SetTranslation(ThisApplication.TransientGeometry.CreateVector(oStep, oStep, 0))

        Dim oOcc As ComponentOccurrence
        oOcc = oOccs.AddiPartMember("C:\temp\iPartFactory.ipt ", oPos, iRow)
    Next
```

Before running the sample, make sure that C:/temp/iPartFactory.ipt exists and that it is an iPart factory. The first line of the C# sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
Sub Main()
    ' Open the factory document invisible.
    Dim oFactoryDoc As PartDocument
    oFactoryDoc = ThisApplication.Documents.Open("C:\temp\iPartFactory.ipt", False)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oFactoryDoc.ComponentDefinition

    ' Make sure we have an iPart factory.
    If oCompDef.IsiPartFactory = False Then
        MsgBox("Chosen document is not a factory.", vbExclamation)
        Exit Sub
    End If

    ' Set a reference to the factory.
    Dim oiPartFactory As iPartFactory
    oiPartFactory = oCompDef.iPartFactory

    ' Get the number of rows in the factory.
    Dim iNumRows As Integer
    iNumRows = oiPartFactory.TableRows.Count

    ' Create a new assembly document
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, , True)

    Dim oOccs As ComponentOccurrences
    oOccs = oDoc.ComponentDefinition.Occurrences

    Dim oPos As Matrix
    oPos = ThisApplication.TransientGeometry.CreateMatrix

    Dim oStep As Double
    oStep = 0#
    Dim iRow As Long

    ' Add an occurrence for each member in the factory.
    For iRow = 1 To iNumRows

        oStep = oStep + 10

        ' Add a translation along X axis
        oPos.SetTranslation(ThisApplication.TransientGeometry.CreateVector(oStep, oStep, 0))

        Dim oOcc As ComponentOccurrence
        oOcc = oOccs.AddiPartMember("C:\temp\iPartFactory.ipt ", oPos, iRow)
    Next
End Sub
```

Assembly Add Occurrence

Description

This sample demonstrates placing an assembly occurrence.

Code Samples

- VBA
- iLogic
- C#

Before running the sample, you need to open an assembly and create a part file called C:\Temp\Part1.ipt, or edit the sample code to point to another part file if desired.

Copy Code

```
Public Sub AddOccurrence()
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create a matrix.  A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    Set oMatrix = oTG.CreateMatrix

    ' Set the rotation of the matrix for a 45 degree rotation about the Z axis.
    Call oMatrix.SetToRotation(3.14159265358979 / 4, _
                           oTG.CreateVector(0, 0, 1), oTG.CreatePoint(0, 0, 0))

    ' Set the translation portion of the matrix so the part will be positioned
    ' at (3,2,1).
    Call oMatrix.SetTranslation(oTG.CreateVector(3, 2, 1))

    ' Add the occurrence.
    Dim oOcc As ComponentOccurrence
    Set oOcc = oAsmCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)
End Sub
```

Before running the sample, you need to open an assembly and create a part file called C:\Temp\Part1.ipt, or edit the sample code to point to another part file if desired.

Copy Code

```
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create a matrix.  A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = oTG.CreateMatrix

    ' Set the rotation of the matrix for a 45 degree rotation about the Z axis.
    Call oMatrix.SetToRotation(3.14159265358979 / 4, _
                           oTG.CreateVector(0, 0, 1), oTG.CreatePoint(0, 0, 0))

    ' Set the translation portion of the matrix so the part will be positioned
    ' at (3,2,1).
    Call oMatrix.SetTranslation(oTG.CreateVector(3, 2, 1))

    ' Add the occurrence.
    Dim oOcc As ComponentOccurrence
    oOcc = oAsmCompDef.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)
```

Before running the sample, you need to open an assembly and create a part file called C:/Temp/Part1.ipt, or edit the sample code to point to another part file if desired. The first line of the C# sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void AddOccurrence()
{
    Application oApp = ThisApplication;

    // Set a reference to the active assembly document.
    // This assumes an assembly document is open.
    AssemblyDocument oDoc = (AssemblyDocument)oApp.ActiveDocument;

    // Set a reference to the assembly component definition.
    AssemblyComponentDefinition oAsmCompDef = oDoc.ComponentDefinition;

    // Set a reference to the transient geometry object.
    TransientGeometry oTG = oApp.TransientGeometry;

    // Create a matrix.  A new matrix is initialized with an identity matrix.
    Matrix oMatrix = oTG.CreateMatrix();

    // Set the rotation of the matrix for a 45 degree rotation about the Z axis.
    oMatrix.SetToRotation(3.14159265358979 / 4,
```

```
        oTG.CreateVector(0, 0, 1), oTG.CreatePoint(0, 0, 0));

    // Set the translation portion of the matrix so the part will be positioned
    // at (3,2,1).
    oMatrix.SetTranslation(oTG.CreateVector(3, 2, 1), true);

    // Add the occurrence.
    ComponentOccurrence oOcc = oAsmCompDef.Occurrences.Add("C:/Temp/Part1.ipt", oMatrix);

}
```

iMate Creation During Occurrence Placement

Description

This sample demonstrates creating multiple iMate results when adding an occurrence into an assembly. This uses the AddUsingiMate method which is the equivalent of using the Place Component command and checking the Use iMate check box on the dialog.

Code Samples

- VBA
- iLogic

To use this sample create a new part by extruding a rectangle to create a cube. Create a mate iMate on one of the faces. Next, create a flush iMate on any of the faces connecting to the first face. Save this part to C:\TempiMatePart.ipt. Finally, have an assembly open and run the sample code.

Copy Code

```
Sub Main()
    ' Get the component definition of the currently open assembly.
    ' This will fail if an assembly document is not open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Create a new matrix object. It will be initialized to an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    ' Place the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Add("C:\TempiMatePart.ipt", oMatrix)

    ' Place the second occurrence, but use iMates for its placement. This is
    ' equivalent to "Use iMate" check box on the "Place Component" dialog.
    Dim oOccEnumerator As ComponentOccurrencesEnumerator
    oOccEnumerator = oAsmCompDef.Occurrences.AddUsingiMates("C:\TempiMatePart.ipt", False)

    ' Since the 'PlaceAllMatching' flag was specified as False, we can be
    ' sure that just one ComponentOccurrence was returned in the enumerator.
    oOcc1 = oOccEnumerator.Item(1)
End Sub
```

To use this sample create a new part by extruding a rectangle to create a cube. Create a mate iMate on one of the faces. Next, create a flush iMate on any of the faces connecting to the first face. Save this part to C:\TempiMatePart.ipt. Finally, have an assembly open and run the sample code.

Copy Code

```
    ' Get the component definition of the currently open assembly.
    ' This will fail if an assembly document is not open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Create a new matrix object. It will be initialized to an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    ' Place the first occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Add("C:\TempiMatePart.ipt", oMatrix)

    ' Place the second occurrence, but use iMates for its placement. This is
    ' equivalent to "Use iMate" check box on the "Place Component" dialog.
    Dim oOccEnumerator As ComponentOccurrencesEnumerator
    oOccEnumerator = oAsmCompDef.Occurrences.AddUsingiMates("C:\TempiMatePart.ipt", False)

    ' Since the 'PlaceAllMatching' flag was specified as False, we can be
    ' sure that just one ComponentOccurrence was returned in the enumerator.
    oOcc1 = oOccEnumerator.Item(1)
```

Traverse an Assembly

Description

This sample shows how to recursively traverse an assembly and get the count of leaf node components and subassemblies.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub AssemblyCount()
    ' Set reference to active document.
    ' This assumes the active document is an assembly
    Dim oDoc As Inventor.AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Get assembly component definition
    Dim oCompDef As Inventor.ComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim sMsg As String
    Dim iLeafNodes As Long
    Dim iSubAssemblies As Long

    ' Get all occurrences from component definition for Assembly document
    Dim oCompOcc As ComponentOccurrence
    For Each oCompOcc In oCompDef.Occurrences
        ' Check if it's child occurrence (leaf node)
        If oCompOcc.SubOccurrences.Count = 0 Then
            Debug.Print oCompOcc.Name
            iLeafNodes = iLeafNodes + 1
        Else
            Debug.Print oCompOcc.Name
            iSubAssemblies = iSubAssemblies + 1
            Call processAllSubOcc(oCompOcc, _
                                  sMsg, _
                                  iLeafNodes, _
                                  iSubAssemblies) ' subassembly
        End If
    Next

    Debug.Print "No of leaf nodes    : " + CStr(iLeafNodes)
    Debug.Print "No of sub assemblies: " + CStr(iSubAssemblies)
End Sub

' This function is called for processing sub assembly.  It is called recursively
' to iterate through the entire assembly tree.
Private Sub processAllSubOcc(ByVal oCompOcc As ComponentOccurrence, _
                             ByRef sMsg As String, _
                             ByRef iLeafNodes As Long, _
                             ByRef iSubAssemblies As Long)

    Dim oSubCompOcc As ComponentOccurrence
    For Each oSubCompOcc In oCompOcc.SubOccurrences
        ' Check if it's child occurrence (leaf node)
        If oSubCompOcc.SubOccurrences.Count = 0 Then
            Debug.Print oSubCompOcc.Name
            iLeafNodes = iLeafNodes + 1
        Else
            sMsg = sMsg + oSubCompOcc.Name + vbCr
            iSubAssemblies = iSubAssemblies + 1

            Call processAllSubOcc(oSubCompOcc, _
                                  sMsg, _
                                  iLeafNodes, _
                                  iSubAssemblies)
        End If
    Next
End Sub
```

Copy Code

```
Sub Main
    ' Set reference to active document.
    ' This assumes the active document is an assembly
    Dim oDoc As Inventor.AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    ' Get assembly component definition
    Dim oCompDef As Inventor.ComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim sMsg As String
    Dim iLeafNodes As Long
    Dim iSubAssemblies As Long

    ' Get all occurrences from component definition for Assembly document
    Dim oCompOcc As ComponentOccurrence
    For Each oCompOcc In oCompDef.Occurrences
        ' Check if it's child occurrence (leaf node)
        If oCompOcc.SubOccurrences.Count = 0 Then
            Logger.Info(oCompOcc.Name)

            iLeafNodes = iLeafNodes + 1
        Else
            Logger.Info(oCompOcc.Name)

            iSubAssemblies = iSubAssemblies + 1
            Call processAllSubOcc(oCompOcc, _
                                  sMsg, _
                                  iLeafNodes, _
                                  iSubAssemblies) ' subassembly
        End If
    Next

    Logger.Info("No of leaf nodes    : " + CStr(iLeafNodes))

    Logger.Info("No of sub assemblies: " + CStr(iSubAssemblies))

End Sub

' This function is called for processing sub assembly.  It is called recursively
' to iterate through the entire assembly tree.
Private Sub processAllSubOcc(ByVal oCompOcc As ComponentOccurrence, _
```

```
                            ByRef sMsg As String, _
                            ByRef iLeafNodes As Long, _
                            ByRef iSubAssemblies As Long)

    Dim oSubCompOcc As ComponentOccurrence
    For Each oSubCompOcc In oCompOcc.SubOccurrences
        ' Check if it's child occurrence (leaf node)
        If oSubCompOcc.SubOccurrences.Count = 0 Then
            Logger.Info(oSubCompOcc.Name)

            iLeafNodes = iLeafNodes + 1
        Else
            sMsg = sMsg + oSubCompOcc.Name + vbCr
            iSubAssemblies = iSubAssemblies + 1

            Call processAllSubOcc(oSubCompOcc, _
                                  sMsg, _
                                  iLeafNodes, _
                                  iSubAssemblies)
        End If
    Next
End Sub
```

Create assembly occurrence with representations

Description

This sample demonstrates how to create an assembly occurrence by specifying various representations.

Code Samples

- VBA
- iLogic

Before running this sample, make sure that the file C:\Temp\Reps.iam exists (or change the path in the sample). The file must contain a model state named MyModelState, a positional representation named MyPositionalRep and a design view representation named MyDesignViewRep.

Copy Code

```
Public Sub AddOccurrenceWithRepresentations()
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create a matrix. A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    Set oMatrix = oTG.CreateMatrix

    ' Create a new NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Set the representations to use when creating the occurrence.
    Call oOptions.Add("ModelState", "MyModelState")
    Call oOptions.Add("PositionalRepresentation", "MyPositionalRep")
    Call oOptions.Add("DesignViewRepresentation", "MyDesignViewRep")
    Call oOptions.Add("DesignViewAssociative", True)

    ' Add the occurrence.
    Dim oOcc As ComponentOccurrence
    Set oOcc = oAsmCompDef.Occurrences.AddWithOptions("C:\Temp\Reps.iam", oMatrix, oOptions)
End Sub
```

Before running this sample, make sure that the file C:\Temp\Reps.iam exists (or change the path in the sample). The file must contain a model state named MyModelState, a positional representation named MyPositionalRep and a design view representation named MyDesignViewRep.

Copy Code

```
    ' Set a reference to the assembly component definintion.
    ' This assumes an assembly document is open.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create a matrix. A new matrix is initialized with an identity matrix.
    Dim oMatrix As Matrix
    oMatrix = oTG.CreateMatrix

    ' Create a new NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Set the representations to use when creating the occurrence.
    Call oOptions.Add("ModelState", "MyModelState")
    Call oOptions.Add("PositionalRepresentation", "MyPositionalRep")
    Call oOptions.Add("DesignViewRepresentation", "MyDesignViewRep")
    Call oOptions.Add("DesignViewAssociative", True)

    ' Add the occurrence.
```

```
    Dim oOcc As ComponentOccurrence
    oOcc = oAsmCompDef.Occurrences.AddWithOptions("C:\Temp\Reps.iam", oMatrix, oOptions)
```

Print instance properties of all components in an assembly

Description

This sample demonstrates how to get the instance properties of all components in an assembly.

Code Samples

- VBA
- iLogic

This VBA sample demonstrates how to get the instance properties of all components in an assembly.

Copy Code

```
Sub PrintInstancePropertiesSample()
    If ThisApplication.ActiveDocument Is Nothing Then
        MsgBox "Please open an assembly document!"
    ElseIf (ThisApplication.ActiveDocument.DocumentType <> kAssemblyDocumentObject) Then
        MsgBox "Please open an assembly document!"
    Else

        Dim oDoc As AssemblyDocument
        Set oDoc = ThisApplication.ActiveDocument

        GetInstancePropInfo oDoc.ComponentDefinition.Occurrences

    End If
End Sub

Sub GetInstancePropInfo(oOccus As ComponentOccurrences)

    Dim oOccu As ComponentOccurrence
    Dim oTempOccu As ComponentOccurrence

    ' The Instance Properties is accessiable via ComponentOccurrence only
    ' so below will get the ComponentOccurrence from ComponentOccurrenceProxy.
    For Each oTempOccu In oOccus
        If oTempOccu.Type = kComponentOccurrenceProxyObject Then
            Set oOccu = oTempOccu.NativeObject

        Else
            Set oOccu = oTempOccu
        End If

        Debug.Print oOccu.Name
        '  Instance Properties
        If oOccu.OccurrencePropertySetsEnabled Then
            Dim oProp As Inventor.Property
            For Each oProp In oOccu.OccurrencePropertySets(1)

                ' Print property info
                Debug.Print "    " & oProp.DisplayName & ":" & oProp.Expression
            Next
        End If

        GetInstancePropInfo oTempOccu.SubOccurrences
    Next
End Sub
```

This VBA sample demonstrates how to get the instance properties of all components in an assembly.

Copy Code

```
Sub Main
    If ThisApplication.ActiveDocument Is Nothing Then
        MsgBox("Please open an assembly document!")
    ElseIf (ThisApplication.ActiveDocument.DocumentType <> kAssemblyDocumentObject) Then
        MsgBox("Please open an assembly document!")
    Else

        Dim oDoc As AssemblyDocument
        oDoc = ThisApplication.ActiveDocument

        GetInstancePropInfo(oDoc.ComponentDefinition.Occurrences)

    End If
End Sub

Sub GetInstancePropInfo(oOccus As ComponentOccurrences)

    Dim oOccu As ComponentOccurrence
    Dim oTempOccu As ComponentOccurrence

    ' The Instance Properties is accessiable via ComponentOccurrence only
    ' so below will get the ComponentOccurrence from ComponentOccurrenceProxy.
    For Each oTempOccu In oOccus
        If oTempOccu.Type = kComponentOccurrenceProxyObject Then
            oOccu = oTempOccu.NativeObject

        Else
            oOccu = oTempOccu
        End If

        Logger.Info(oOccu.Name)
```

```
        '  Instance Properties
        If oOccu.OccurrencePropertySetsEnabled Then
            Dim oProp As Inventor.Property
            For Each oProp In oOccu.OccurrencePropertySets(1)

                ' Print property info
                Logger.Info("    " & oProp.DisplayName & ":" & oProp.Expression)

            Next
        End If

        GetInstancePropInfo(oTempOccu.SubOccurrences)
    Next
End Sub
```

Assembly Move Occurrence

Description

This sample demonstrates moving a component occurrence. This sample performs a translate, but a rotate can also be performed since the transform is defined using a matrix.

Code Samples

- VBA
- iLogic

Before running the sample you need to open an assembly and select the occurrence to move. The sample code first moves the occurrence honoring any existing constraints and then moves it ignoring any constraints.

Copy Code

```
Public Sub MoveOccurrence()
   ' Set a reference to the assembly component definintion.
   Dim oAsmCompDef As AssemblyComponentDefinition
   Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

   ' Get an occurrence from the select set.
   On Error Resume Next
   Dim oOccurrence As ComponentOccurrence
   Set oOccurrence = ThisApplication.ActiveDocument.SelectSet.Item(1)
   If Err Then
      MsgBox "An occurrence must be selected."
      Exit Sub
   End If
   On Error GoTo 0

   ' Get the current transformation matrix from the occurrence.
   Dim oTransform As Matrix
   Set oTransform = oOccurrence.Transformation

   ' Move the occurrence honoring any existing constraints.
   oTransform.SetTranslation ThisApplication.TransientGeometry.CreateVector(2, 2, 3)
   oOccurrence.Transformation = oTransform

   ' Move the occurrence ignoring any constraints.
   ' Anything that causes the assembly to recompute will cause the
   ' occurrence to reposition itself to honor the constraints.
   oTransform.SetTranslation ThisApplication.TransientGeometry.CreateVector(3, 4, 5)
   Call oOccurrence.SetTransformWithoutConstraints(oTransform)
End Sub
```

Before running the sample you need to open an assembly and select the occurrence to move. The sample code first moves the occurrence honoring any existing constraints and then moves it ignoring any constraints.

Copy Code

```
    ' Set a reference to the assembly component definintion.
   Dim oAsmCompDef As AssemblyComponentDefinition
   oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

   ' Get an occurrence from the select set.
   On Error Resume Next
   Dim oOccurrence As ComponentOccurrence
   oOccurrence = ThisApplication.ActiveDocument.SelectSet.Item(1)
   If Err.Number > 0 Then
      MsgBox("An occurrence must be selected.")
      Exit Sub
   End If
   On Error GoTo 0

   ' Get the current transformation matrix from the occurrence.
   Dim oTransform As Matrix
   oTransform = oOccurrence.Transformation

   ' Move the occurrence honoring any existing constraints.
   oTransform.SetTranslation(ThisApplication.TransientGeometry.CreateVector(2, 2, 3))
   oOccurrence.Transformation = oTransform

   ' Move the occurrence ignoring any constraints.
   ' Anything that causes the assembly to recompute will cause the
   ' occurrence to reposition itself to honor the constraints.
   oTransform.SetTranslation(ThisApplication.TransientGeometry.CreateVector(3, 4, 5))
   Call oOccurrence.SetTransformWithoutConstraints(oTransform)
```

Create Approximate Polyline to 3D Curve

Description

Draws a polyline that is an approximation of the selected curve. To use this have a part open that contains a 3D skech that contains curves.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub Approximate3DSketchGeometry()
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Have the user select a sketch entity.
    Dim selectObj As SketchEntity3D
    Set selectObj = ThisApplication.CommandManager.Pick(kSketch3DCurveFilter, "Select 3D sketch entity")
    If selectObj Is Nothing Then
        On Error Resume Next
        Call partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test").Delete
        Call partDoc.GraphicsDataSetsCollection.Item("Test").Delete
        ThisApplication.ActiveView.Update
        Exit Sub
    End If

    ' Get the tolerance to approximate with.
    Dim tolerance As Double
    tolerance = Val(InputBox("Enter the chord height tolerance:", "Tolerance", "0.25"))

    ' Get the evaluator from the curve.
    Dim eval As CurveEvaluator
    Set eval = selectObj.Geometry.Evaluator

    ' Get the parameter extents.
    Dim startParam As Double
    Dim endParam As Double
    Call eval.GetParamExtents(startParam, endParam)

    Dim vertexCount As Long
    Dim vertexCoords() As Double
    Call eval.GetStrokes(startParam, endParam, tolerance, vertexCount, vertexCoords)

    ' Create a client graphics object.  If one already exists, give the user
    ' the option of re-using it, or creating a new one.
    Dim graphics As ClientGraphics
    Dim graphicsData As GraphicsDataSets
    On Error Resume Next
    Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test")
    On Error GoTo 0
    If graphics Is Nothing Then
        Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
        Set graphicsData = partDoc.GraphicsDataSetsCollection.Add("Test")
    Else
        Dim answer As VbMsgBoxResult
        answer = MsgBox("Yes to add to existing graphics. No to create new graphics. Cancel to clean graphics and quit.", vbYesNoCance
        If answer = vbNo Then
            On Error Resume Next
            graphics.Delete
            partDoc.GraphicsDataSetsCollection.Item("Test").Delete
            On Error GoTo 0

            Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
            Set graphicsData = partDoc.GraphicsDataSetsCollection.Add("Test")
        ElseIf answer = vbYes Then
            Set graphicsData = partDoc.GraphicsDataSetsCollection.Item("Test")
        ElseIf answer = vbCancel Then
            If Not graphics Is Nothing Then
                graphics.Delete
                partDoc.GraphicsDataSetsCollection.Item("Test").Delete
                ThisApplication.ActiveView.Update
                Exit Sub
            End If
        End If
    End If

    Dim coordSet As GraphicsCoordinateSet
    Set coordSet = graphicsData.CreateCoordinateSet(1)
    Call coordSet.PutCoordinates(vertexCoords)

    ' Create a graphics node.
    Dim node As GraphicsNode
    Set node = graphics.AddNode(1)

    ' Create a line strip using the calculated coordinates.
    Dim lineStrip As LineStripGraphics
    Set lineStrip = node.AddLineStripGraphics
    lineStrip.CoordinateSet = coordSet

    ThisApplication.ActiveView.Update
End Sub
```

To use this sample a part must be active.

Copy Code

```
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument
```

```
    ' Have the user select a sketch entity.
    Dim selectObj As SketchEntity3D
    selectObj = ThisApplication.CommandManager.Pick(kSketch3DCurveFilter, "Select 3D sketch entity")
    If selectObj Is Nothing Then
        On Error Resume Next
        Call partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test").Delete
        Call partDoc.GraphicsDataSetsCollection.Item("Test").Delete
        ThisApplication.ActiveView.Update
        Exit Sub
    End If

    ' Get the tolerance to approximate with.
    Dim tolerance As Double
    tolerance = Val(InputBox("Enter the chord height tolerance:", "Tolerance", "0.25"))

    ' Get the evaluator from the curve.
    Dim eval As CurveEvaluator
    eval = selectObj.Geometry.Evaluator

    ' Get the parameter extents.
    Dim startParam As Double
    Dim endParam As Double
    Call eval.GetParamExtents(startParam, endParam)

    Dim vertexCount As Long
    Dim vertexCoords() As Double = New Double() {}
    Call eval.GetStrokes(startParam, endParam, tolerance, vertexCount, vertexCoords)

    ' Create a client graphics object.  If one already exists, give the user
    ' the option of re-using it, or creating a new one.
    Dim graphics As ClientGraphics
    Dim graphicsData As GraphicsDataSets
    On Error Resume Next
    graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test")
    On Error GoTo 0
    If graphics Is Nothing Then
        graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
        graphicsData = partDoc.GraphicsDataSetsCollection.Add("Test")
    Else
        Dim answer As MsgBoxResult
        answer = MsgBox("Yes to add to existing graphics. No to create new graphics. Cancel to clean graphics and quit.", vbYesNoCance
        If answer = vbNo Then
            On Error Resume Next
            graphics.Delete
            partDoc.GraphicsDataSetsCollection.Item("Test").Delete
            On Error GoTo 0

            graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
            graphicsData = partDoc.GraphicsDataSetsCollection.Add("Test")
        ElseIf answer = vbYes Then
            graphicsData = partDoc.GraphicsDataSetsCollection.Item("Test")
        ElseIf answer = vbCancel Then
            If Not graphics Is Nothing Then
                graphics.Delete
                partDoc.GraphicsDataSetsCollection.Item("Test").Delete
                ThisApplication.ActiveView.Update
                Exit Sub
            End If
        End If
    End If

    Dim coordSet As GraphicsCoordinateSet
    coordSet = graphicsData.CreateCoordinateSet(1)
    Call coordSet.PutCoordinates(vertexCoords)

    ' Create a graphics node.
    Dim node As GraphicsNode
    node = graphics.AddNode(1)

    ' Create a line strip using the calculated coordinates.
    Dim lineStrip As LineStripGraphics
    lineStrip = node.AddLineStripGraphics
    lineStrip.CoordinateSet = coordSet

    ThisApplication.ActiveView.Update
```

Extract a Partial Curve from a Curve

Description

Demonstrates the ability to extract partial curves from a transient geometry curve. This sample has you select an existing sketch spline and extracts three curves from the curve. It then creates real curves using the extracted curves and deletes the original sketch curve.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub ExtractPartialCurves2D()
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    Dim spline As Object
```

```
    Set spline = ThisApplication.CommandManager.Pick(kSketchCurveSplineFilter, "Select 2d spline")

    ' Get the spline geometry from the entity.
    Dim splineCurve As BSplineCurve2d
    Set splineCurve = spline.Geometry

    ' Get the parameter bounds of the curve.
    Dim startParam As Double
    Dim endParam As Double
    Call splineCurve.Evaluator.GetParamExtents(startParam, endParam)

    ' Extract three curves where they are in thirds of the original
    ' relative to the curves parameter space.
    Dim curves(2) As BSplineCurve2d
    Set curves(0) = splineCurve.ExtractPartial(startParam, startParam + (endParam - startParam) / 3)
    Set curves(1) = splineCurve.ExtractPartial(startParam + (endParam - startParam) / 3, startParam + 2 * (endParam - startParam) / 3)
    Set curves(2) = splineCurve.ExtractPartial(startParam + 2 * (endParam - startParam) / 3, endParam)

    ' Create new sketch curves using the extracted splines.
    Dim splineSketch As Sketch
    Set splineSketch = spline.Parent
    Call splineSketch.SketchFixedSplines.Add(curves(0))
    Call splineSketch.SketchFixedSplines.Add(curves(1))
    Call splineSketch.SketchFixedSplines.Add(curves(2))

    ' Delete the original curve.
    spline.Delete
End Sub
```

To use this sample a part must be active.

Copy Code

```
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    Dim spline As Object
    spline = ThisApplication.CommandManager.Pick(kSketchCurveSplineFilter, "Select 2d spline")

    ' Get the spline geometry from the entity.
    Dim splineCurve As BSplineCurve2d
    splineCurve = spline.Geometry

    ' Get the parameter bounds of the curve.
    Dim startParam As Double
    Dim endParam As Double
    Call splineCurve.Evaluator.GetParamExtents(startParam, endParam)

    ' Extract three curves where they are in thirds of the original
    ' relative to the curves parameter space.
    Dim curves(2) As BSplineCurve2d
    curves(0) = splineCurve.ExtractPartial(startParam, startParam + (endParam - startParam) / 3)
    curves(1) = splineCurve.ExtractPartial(startParam + (endParam - startParam) / 3, startParam + 2 * (endParam - startParam) / 3)
    curves(2) = splineCurve.ExtractPartial(startParam + 2 * (endParam - startParam) / 3, endParam)

    ' Create new sketch curves using the extracted splines.
    Dim splineSketch As sketch
    splineSketch = spline.Parent
    Call splineSketch.SketchFixedSplines.Add(curves(0))
    Call splineSketch.SketchFixedSplines.Add(curves(1))
    Call splineSketch.SketchFixedSplines.Add(curves(2))

    ' Delete the original curve.
    spline.Delete
```

Body Imprinting and matching the results

Description

This sample is intended to demonstrate a technique of finding the matching surfaces between the original input bodies and output imprinted bodies. This relies on transient keys, which is a unique ID associated with each B-Rep entity. A transient key is only good as long as the model is not recomputed.

Code Samples

- VBA
- iLogic

To use this sample you need to have a part with multiple bodies. Two of the bodies should be coincident and those two bodies can be selected when running the sample.

Copy Code

```
Public Sub SampleImprintMatching2()
    ' Get the active assembly document and its definition.
    Dim doc As PartDocument
    Set doc = ThisApplication.ActiveDocument
    Dim def As PartComponentDefinition
    Set def = doc.ComponentDefinition

    ' Have two parts selected in the assembly.
    Dim part1 As ComponentOccurrence
    Dim part2 As ComponentOccurrence
    Dim origBody1 As SurfaceBody
    Dim origBody2 As SurfaceBody
    Set origBody1 = ThisApplication.CommandManager.Pick(kPartBodyFilter, "Select body 1")
    Set origBody2 = ThisApplication.CommandManager.Pick(kPartBodyFilter, "Select body 2")

    ' Create alternate bodies.  Ideally this wouldn't be needed but this adds some special
```

```
' tags to the bodies that allow transient keys to continue to work after the ImprintBodies method is called.
' After this there is the original two bodies and the alternate version of each of the
' bodies.  The alternate may have more faces than the original, but the transient keys
' between the two should still match.
Dim body1 As SurfaceBody
Dim body2 As SurfaceBody
Set body1 = origBody1.AlternateBody(Inventor.SurfaceGeometryFormEnum.SurfaceGeometryForm_NURBS)
Set body2 = origBody2.AlternateBody(Inventor.SurfaceGeometryFormEnum.SurfaceGeometryForm_NURBS)

' Imprint the bodies.  After this call we now have three sets of bodies; the original, the alternate,
' and the imprinted bodies.  There should be additional faces in the imprinted bodies but the
' transient keys should still match for all three sets of bodies.
Dim newBody1 As SurfaceBody
Dim newBody2 As SurfaceBody
Dim body1OverlapFaces As Faces
Dim body2OverlapFaces As Faces
Dim body1OverlapEdges As Edges
Dim body2OverlapEdges As Edges
Call ThisApplication.TransientBRep.ImprintBodies(body1, body2, True, newBody1, newBody2, body1OverlapFaces, body2OverlapFaces, bod

' The code below creates new non-parametric base features using the new imprinted bodies
' so that the results can be visualized.  The new bodies are created offset from the
' original so that they don't overlap and are more easily seen.
'
' ** After this operation there will be four sets of bodies; the original, the alternate form,
' ** the imprinted bodies, and now the body created as a result of the feature creation.
' ** Because a new persisted body is created as part of the operation, new transient keys
' ** are assigned to the body, so the transient keys will no longer match this body.
' ** However, because they're essentially just copies of the imprinted bodies, the faces
' ** are in the same order in the two bodies so the indexes can be used to match between
' ** the two bodies.

' Define a matrix to use in transforming the bodies.
Dim trans As Matrix
Set trans = ThisApplication.TransientGeometry.CreateMatrix

' Move the first imprinted body over based on the range so it doesn't lie on the original.
' Moving the body is a modification of the existing body and leaves the transient keys as-is.
' However, it does change the face identities so comparing IUnknowns will no longer work.  Because
' of this I first save the indices of the overlapping faces while the faces are the same.
    Dim matchingIndexList1() As Integer
Dim matchingIndexList2() As Integer
ReDim matchingIndexList1(body1OverlapFaces.Count - 1)
ReDim matchingIndexList2(body2OverlapFaces.Count - 1)
Dim i As Integer
For i = 1 To body1OverlapFaces.Count
    ' Find the indices of the overlapping face in the Faces collection.
    Dim j As Integer
    For j = 1 To newBody1.Faces.Count
        If body1OverlapFaces.Item(i) Is newBody1.Faces.Item(j) Then
            matchingIndexList1(i - 1) = j
            Exit For
        End If
    Next

    Dim body2Index As Integer
    For j = 1 To newBody2.Faces.Count
        If body2OverlapFaces.Item(i) Is newBody2.Faces.Item(j) Then
            matchingIndexList2(i - 1) = j
            Exit For
        End If
    Next
Next

' Now do the transformation of the first body.
trans.Cell(1, 4) = (body1.RangeBox.MaxPoint.X - body1.RangeBox.MinPoint.X) * 1.1
Call ThisApplication.TransientBRep.Transform(newBody1, trans)

' Move the second imprinted body over based on the range so it doesn't lie on the original.
trans.Cell(1, 4) = trans.Cell(1, 4) + (body1.RangeBox.MaxPoint.X - body1.RangeBox.MinPoint.X) * 1.1
Call ThisApplication.TransientBRep.Transform(newBody2, trans)

' Create a new feature from the first imprinted body.
Dim nonParaDef As NonParametricBaseFeatureDefinition
Set nonParaDef = def.Features.NonParametricBaseFeatures.CreateDefinition
Dim bodyColl As ObjectCollection
Set bodyColl = ThisApplication.TransientObjects.CreateObjectCollection
Call bodyColl.Add(newBody1)
nonParaDef.BRepEntities = bodyColl
nonParaDef.OutputType = kSolidOutputType
Dim non1 As NonParametricBaseFeature
Set non1 = def.Features.NonParametricBaseFeatures.AddByDefinition(nonParaDef)
Dim resultBody1 As SurfaceBody
Set resultBody1 = non1.SurfaceBodies.Item(1)

' Create a new feature from the second imprinted body.
Set nonParaDef = def.Features.NonParametricBaseFeatures.CreateDefinition
Set bodyColl = ThisApplication.TransientObjects.CreateObjectCollection
Call bodyColl.Add(newBody2)
nonParaDef.BRepEntities = bodyColl
nonParaDef.OutputType = kSolidOutputType
Dim non2 As NonParametricBaseFeature
Set non2 = def.Features.NonParametricBaseFeatures.AddByDefinition(nonParaDef)
Dim resultBody2 As SurfaceBody
Set resultBody2 = non2.SurfaceBodies.Item(1)

' Fit the view to see the result.
Dim cam As Camera
Set cam = ThisApplication.ActiveView.Camera
cam.Fit
cam.Apply

' Create a highlight set used to see the matches.
Dim hs As HighlightSet
Set hs = doc.CreateHighlightSet

' Show the matches between the first imprint body and the original body.
```

```
    For i = 1 To newBody1.Faces.Count
        ' The face in the feature has a different transient key so get the
        ' face as the same index and assume it's the same.
        Dim newFace As Face
        Set newFace = resultBody1.Faces.Item(i)

        hs.AddItem newFace

        ' Get the corresponding face in the original body using the transient key from
        ' the imprinted face at the current index in the face collection.
        Dim otherFace As Face
        Set otherFace = origBody1.BindTransientKeyToObject(newBody1.Faces.Item(i).TransientKey)

        hs.AddItem otherFace

        MsgBox "Match"
        hs.Clear
    Next

    ' Show the matches between the second imprint body and the original body.
    For i = 1 To newBody2.Faces.Count
        ' The face in the feature has a different transient key so get the
        ' face as the same index and assume it's the same.
        Set newFace = resultBody2.Faces.Item(i)

        hs.AddItem newFace

        ' Get the corresponding face in the original body using the transient key from
        ' the imprinted face at the current index in the face collection.
        Set otherFace = origBody2.BindTransientKeyToObject(newBody2.Faces.Item(i).TransientKey)

        hs.AddItem otherFace

        MsgBox "Match"
        hs.Clear
    Next

    ' Highlight just the overlapping faces.
    ' The collection of overlapping faces returned by the ImprintBodies method correspond to
    ' the bodies returned by the ImprintBodies function.  This means the transient keys on the
    ' overlapping bodies will match all of the bodies except for the bodies created by the
    ' non-parametric base features.  The face index was save previously and will be used here.
    For i = 1 To body1OverlapFaces.Count
        Dim overlapFace1 As Face
        Dim overlapFace2 As Face
        Set overlapFace1 = resultBody1.Faces.Item(matchingIndexList1(i - 1))
        Set overlapFace2 = resultBody2.Faces.Item(matchingIndexList2(i - 1))

        hs.AddItem overlapFace1
        hs.AddItem overlapFace2

        MsgBox "Overlap Match"
        hs.Clear
    Next
End Sub
```

To use this sample you need to have a part with multiple bodies. Two of the bodies should be coincident and those two bodies can be selected when running the sample.

Copy Code

```
    ' Get the active assembly document and its definition.
    Dim doc As PartDocument
    doc = ThisApplication.ActiveDocument
    Dim def As PartComponentDefinition
    def = doc.ComponentDefinition

    ' Have two parts selected in the assembly.
    Dim part1 As ComponentOccurrence
    Dim part2 As ComponentOccurrence
    Dim origBody1 As SurfaceBody
    Dim origBody2 As SurfaceBody
    origBody1 = ThisApplication.CommandManager.Pick(kPartBodyFilter, "Select body 1")
    origBody2 = ThisApplication.CommandManager.Pick(kPartBodyFilter, "Select body 2")

    ' Create alternate bodies.  Ideally this wouldn't be needed but this adds some special
    ' tags to the bodies that allow transient keys to continue to work after the ImprintBodies method is called.
    ' After this there is the original two bodies and the alternate version of each of the
    ' bodies.  The alternate may have more faces than the original, but the transient keys
    ' between the two should still match.
    Dim body1 As SurfaceBody
    Dim body2 As SurfaceBody
    body1 = origBody1.AlternateBody(Inventor.SurfaceGeometryFormEnum.SurfaceGeometryForm_NURBS)
    body2 = origBody2.AlternateBody(Inventor.SurfaceGeometryFormEnum.SurfaceGeometryForm_NURBS)

    ' Imprint the bodies.  After this call we now have three sets of bodies; the original, the alternate,
    ' and the imprinted bodies.  There should be additional faces in the imprinted bodies but the
    ' transient keys should still match for all three sets of bodies.
    Dim newBody1 As SurfaceBody
    Dim newBody2 As SurfaceBody
    Dim body1OverlapFaces As Faces
    Dim body2OverlapFaces As Faces
    Dim body1OverlapEdges As Edges
    Dim body2OverlapEdges As Edges
    Call ThisApplication.TransientBRep.ImprintBodies(body1, body2, True, newBody1, newBody2, body1OverlapFaces, body2OverlapFaces, bod

    ' The code below creates new non-parametric base features using the new imprinted bodies
    ' so that the results can be visualized.  The new bodies are created offset from the
    ' original so that they don't overlap and are more easily seen.
    '
    ' ** After this operation there will be four sets of bodies; the original, the alternate form,
    ' ** the imprinted bodies, and now the body created as a result of the feature creation.
    ' ** Because a new persisted body is created as part of the operation, new transient keys
    ' ** are assigned to the body, so the transient keys will no longer match this body.
    ' ** However, because they're essentially just copies of the imprinted bodies, the faces
    ' ** are in the same order in the two bodies so the indexes can be used to match between
    ' ** the two bodies.
```

```
' Define a matrix to use in transforming the bodies.
Dim trans As Matrix
trans = ThisApplication.TransientGeometry.CreateMatrix

' Move the first imprinted body over based on the range so it doesn't lie on the original.
' Moving the body is a modification of the existing body and leaves the transient keys as-is.
' However, it does change the face identities so comparing IUnknowns will no longer work.  Because
' of this I first save the indices of the overlapping faces while the faces are the same.
    Dim matchingIndexList1() As Integer
Dim matchingIndexList2() As Integer
ReDim matchingIndexList1(body1OverlapFaces.Count - 1)
ReDim matchingIndexList2(body2OverlapFaces.Count - 1)
Dim i As Integer
For i = 1 To body1OverlapFaces.Count
    ' Find the indices of the overlapping face in the Faces collection.
    Dim j As Integer
    For j = 1 To newBody1.Faces.Count
        If body1OverlapFaces.Item(i) Is newBody1.Faces.Item(j) Then
            matchingIndexList1(i - 1) = j
            Exit For
        End If
    Next

    Dim body2Index As Integer
    For j = 1 To newBody2.Faces.Count
        If body2OverlapFaces.Item(i) Is newBody2.Faces.Item(j) Then
            matchingIndexList2(i - 1) = j
            Exit For
        End If
    Next
Next

' Now do the transformation of the first body.
trans.Cell(1, 4) = (body1.RangeBox.MaxPoint.X - body1.RangeBox.MinPoint.X) * 1.1
Call ThisApplication.TransientBRep.Transform(newBody1, trans)

' Move the second imprinted body over based on the range so it doesn't lie on the original.
trans.Cell(1, 4) = trans.Cell(1, 4) + (body1.RangeBox.MaxPoint.X - body1.RangeBox.MinPoint.X) * 1.1
Call ThisApplication.TransientBRep.Transform(newBody2, trans)

' Create a new feature from the first imprinted body.
Dim nonParaDef As NonParametricBaseFeatureDefinition
nonParaDef = def.Features.NonParametricBaseFeatures.CreateDefinition
Dim bodyColl As ObjectCollection
bodyColl = ThisApplication.TransientObjects.CreateObjectCollection
Call bodyColl.Add(newBody1)
nonParaDef.BRepEntities = bodyColl
nonParaDef.OutputType = kSolidOutputType
Dim non1 As NonParametricBaseFeature
non1 = def.Features.NonParametricBaseFeatures.AddByDefinition(nonParaDef)
Dim resultBody1 As SurfaceBody
resultBody1 = non1.SurfaceBodies.Item(1)

' Create a new feature from the second imprinted body.
nonParaDef = def.Features.NonParametricBaseFeatures.CreateDefinition
bodyColl = ThisApplication.TransientObjects.CreateObjectCollection
Call bodyColl.Add(newBody2)
nonParaDef.BRepEntities = bodyColl
nonParaDef.OutputType = kSolidOutputType
Dim non2 As NonParametricBaseFeature
non2 = def.Features.NonParametricBaseFeatures.AddByDefinition(nonParaDef)
Dim resultBody2 As SurfaceBody
resultBody2 = non2.SurfaceBodies.Item(1)

' Fit the view to see the result.
Dim cam As Camera
cam = ThisApplication.ActiveView.Camera
cam.Fit
cam.Apply

' Create a highlight set used to see the matches.
Dim hs As HighlightSet
hs = doc.CreateHighlightSet

Dim newFace As Face
Dim otherFace As Face

' Show the matches between the first imprint body and the original body.
For i = 1 To newBody1.Faces.Count
    ' The face in the feature has a different transient key so get the
    ' face as the same index and assume it's the same.

    newFace = resultBody1.Faces.Item(i)

    hs.AddItem(newFace)

    ' Get the corresponding face in the original body using the transient key from
    ' the imprinted face at the current index in the face collection.
    otherFace = origBody1.BindTransientKeyToObject(newBody1.Faces.Item(i).TransientKey)

    hs.AddItem(otherFace)

    MsgBox("Match")
    hs.Clear
Next

' Show the matches between the second imprint body and the original body.
For i = 1 To newBody2.Faces.Count
    ' The face in the feature has a different transient key so get the
    ' face as the same index and assume it's the same.
    newFace = resultBody2.Faces.Item(i)

    hs.AddItem(newFace)

    ' Get the corresponding face in the original body using the transient key from
    ' the imprinted face at the current index in the face collection.
```

```
        otherFace = origBody2.BindTransientKeyToObject(newBody2.Faces.Item(i).TransientKey)

        hs.AddItem(otherFace)

        MsgBox("Match")
        hs.Clear
    Next

    ' Highlight just the overlapping faces.
    ' The collection of overlapping faces returned by the ImprintBodies method correspond to
    ' the bodies returned by the ImprintBodies function.  This means the transient keys on the
    ' overlapping bodies will match all of the bodies except for the bodies created by the
    ' non-parametric base features.  The face index was save previously and will be used here.
    For i = 1 To body1OverlapFaces.Count
        Dim overlapFace1 As Face
        Dim overlapFace2 As Face
        overlapFace1 = resultBody1.Faces.Item(matchingIndexList1(i - 1))
        overlapFace2 = resultBody2.Faces.Item(matchingIndexList2(i - 1))

        hs.AddItem(overlapFace1)
        hs.AddItem(overlapFace2)

        MsgBox("Overlap Match")
        hs.Clear
    Next
```

Imprint bodies within an assembly.

Description

This sample demonstrates creating imprinted bodies from two selected occurrences in an assembly.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateImprintFromAssembly()
    ' Get the active assembly document and its definition.
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.ActiveDocument

    ' Have two parts selected in the assembly.
    Dim part1 As ComponentOccurrence
    Dim part2 As ComponentOccurrence
    Set part1 = ThisApplication.CommandManager.Pick(kAssemblyLeafOccurrenceFilter, "Select part 1")
    Set part2 = ThisApplication.CommandManager.Pick(kAssemblyLeafOccurrenceFilter, "Select part 2")

    ' Get the bodies associated with the occurrences.  Because of a problem when
    ' this sample was written the ImprintBodies method fails when used with SurfaceBodyProxy
    ' objects.  The code below works around this by creating new transformed bodies that
    ' will provide the equivalent result.

    ' Get the bodies in part space as transient bodies.
    Dim transBrep As TransientBRep
    Set transBrep = ThisApplication.TransientBRep
    Dim body1 As SurfaceBody
    Dim body2 As SurfaceBody
    Set body1 = transBrep.Copy(part1.Definition.SurfaceBodies.Item(1))
    Set body2 = transBrep.Copy(part2.Definition.SurfaceBodies.Item(1))

    ' Transform the bodies to be in the location represented in the assembly.
    Call transBrep.Transform(body1, part1.Transformation)
    Call transBrep.Transform(body2, part2.Transformation)

    ' Imprint the bodies.
    Dim newBody1 As SurfaceBody
    Dim newBody2 As SurfaceBody
    Dim body1OverlapFaces As Faces
    Dim body2OverlapFaces As Faces
    Dim body1OverlapEdges As Edges
    Dim body2OverlapEdges As Edges
    Call ThisApplication.TransientBRep.ImprintBodies(body1, body2, True, newBody1, newBody2, _
                                                     body1OverlapFaces, body2OverlapFaces, _
                                                     body1OverlapEdges, body2OverlapEdges)

    ' Create a new part document to show the resulting bodies in.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                  ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a new feature from the first imprinted body.
    Dim non1 As NonParametricBaseFeature
    Set non1 = partDef.Features.NonParametricBaseFeatures.Add(newBody1)
    Set newBody1 = non1.SurfaceBodies.Item(1)

    ' Create a new feature from the second imprinted body.
    Dim non2 As NonParametricBaseFeature
    Set non2 = partDef.Features.NonParametricBaseFeatures.Add(newBody2)
    Set newBody2 = non2.SurfaceBodies.Item(1)
End Sub
```

Copy Code

```
    ' Get the active assembly document and its definition.
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.ActiveDocument

    ' Have two parts selected in the assembly.
    Dim part1 As ComponentOccurrence
    Dim part2 As ComponentOccurrence
    part1 = ThisApplication.CommandManager.Pick(kAssemblyLeafOccurrenceFilter, "Select part 1")
    part2 = ThisApplication.CommandManager.Pick(kAssemblyLeafOccurrenceFilter, "Select part 2")

    ' Get the bodies associated with the occurrences.  Because of a problem when
    ' this sample was written the ImprintBodies method fails when used with SurfaceBodyProxy
    ' objects.  The code below works around this by creating new transformed bodies that
    ' will provide the equivalent result.

    ' Get the bodies in part space as transient bodies.
    Dim transBrep As TransientBRep
    transBrep = ThisApplication.TransientBRep
    Dim body1 As SurfaceBody
    Dim body2 As SurfaceBody
    body1 = transBrep.Copy(part1.Definition.SurfaceBodies.Item(1))
    body2 = transBrep.Copy(part2.Definition.SurfaceBodies.Item(1))

    ' Transform the bodies to be in the location represented in the assembly.
    Call transBrep.Transform(body1, part1.Transformation)
    Call transBrep.Transform(body2, part2.Transformation)

    ' Imprint the bodies.
    Dim newBody1 As SurfaceBody
    Dim newBody2 As SurfaceBody
    Dim body1OverlapFaces As Faces
    Dim body2OverlapFaces As Faces
    Dim body1OverlapEdges As Edges
    Dim body2OverlapEdges As Edges
    Call ThisApplication.TransientBRep.ImprintBodies(body1, body2, True, newBody1, newBody2, _
                                                     body1OverlapFaces, body2OverlapFaces, _
                                                     body1OverlapEdges, body2OverlapEdges)

    ' Create a new part document to show the resulting bodies in.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
              ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a new feature from the first imprinted body.
    Dim non1 As NonParametricBaseFeature
    non1 = partDef.Features.NonParametricBaseFeatures.Add(newBody1)
    newBody1 = non1.SurfaceBodies.Item(1)

    ' Create a new feature from the second imprinted body.
    Dim non2 As NonParametricBaseFeature
    non2 = partDef.Features.NonParametricBaseFeatures.Add(newBody2)
    newBody2 = non2.SurfaceBodies.Item(1)
```

3D Curve from Parametric Curve

Description

Demonstrates the conversion of a 2d parameteric space curve into the equivalent 3d model space curve. To use this sample you must have a part open. You can select any face and 3D curves will be drawn on the face using client graphics.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub ParamCurveto3D()
    ' Have the user select a face whose parametric space will be used.
    Dim testFace As Face
    Set testFace = ThisApplication.CommandManager.Pick(kPartFaceFilter, "Select a face")

    ' Get the surface evaluator from the face.
    Dim surfEval As SurfaceEvaluator
    Set surfEval = testFace.Evaluator

    ' Get the parametric range of the face.
    Dim paramRange As Box2d
    Set paramRange = surfEval.ParamRangeRect

    ' Get the transient geometry object.
    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Get the active document.  This assumes a part is active.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Create a client graphics object.  If one already exists, give the user
    ' the option of re-using it, or creating a new one.
    Dim graphics As ClientGraphics
    On Error Resume Next
    Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test")
    If Err Then
```

```
    Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
Else
    If MsgBox("Yes to add to existing graphics. No to start over.", vbYesNo + vbQuestion) = vbNo Then
        graphics.Delete

        Set graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
    End If
End If

' Create a new graphics node.
Dim node As GraphicsNode
Set node = graphics.AddNode(1)

' Do a loop that creates 5 lines that go from the minimum range point to
' five points along the maximum u parameter.
Dim startPnt As Point2d
Set startPnt = paramRange.MinPoint
Dim endPnt As Point2d
Dim i As Integer
For i = 1 To 5
    ' Create the 2d line in parametric space.
    Set endPnt = tg.CreatePoint2d(paramRange.MaxPoint.X, (((paramRange.MaxPoint.y - paramRange.MinPoint.y) / 4) * (i - 1)) + paramR
    Dim surfCurve As LineSegment2d
    Set surfCurve = tg.CreateLineSegment2d(startPnt, endPnt)

    ' Get the equivalent 3d curve in model space.
    Dim resultCurve As Object
    Set resultCurve = surfEval.Get3dCurveFrom2dCurve(surfCurve)

    ' Create client graphics for this curve.
    Dim crvGraphics As CurveGraphics
    Set crvGraphics = node.AddCurveGraphics(resultCurve)
    crvGraphics.DepthPriority = 2
Next

' Create a 2d circle in parametric space.
Dim center As Point2d
Set center = tg.CreatePoint2d((paramRange.MaxPoint.X + paramRange.MinPoint.X) / 2, (paramRange.MaxPoint.y + paramRange.MinPoint.y)
Dim radius As Double
radius = (paramRange.MaxPoint.X - paramRange.MinPoint.X) * 0.25
Dim paramCircle As Circle2d
Set paramCircle = tg.CreateCircle2d(center, radius)

' Get the equivalent 3d curve in model space.
Set resultCurve = surfEval.Get3dCurveFrom2dCurve(paramCircle)

' Create client graphics for this curve.
Set crvGraphics = node.AddCurveGraphics(resultCurve)
crvGraphics.DepthPriority = 2

ThisApplication.ActiveView.Update
ThisApplication.ActiveView.Fit
End Sub
```

To use this sample a part must be active.

Copy Code

```
' Have the user select a face whose parametric space will be used.
Dim testFace As Face
testFace = ThisApplication.CommandManager.Pick(kPartFaceFilter, "Select a face")

' Get the surface evaluator from the face.
Dim surfEval As SurfaceEvaluator
surfEval = testFace.Evaluator

' Get the parametric range of the face.
Dim paramRange As Box2d
paramRange = surfEval.ParamRangeRect

' Get the transient geometry object.
Dim tg As TransientGeometry
tg = ThisApplication.TransientGeometry

' Get the active document.  This assumes a part is active.
Dim partDoc As PartDocument
partDoc = ThisApplication.ActiveDocument

' Create a client graphics object.  If one already exists, give the user
' the option of re-using it, or creating a new one.
Dim graphics As ClientGraphics
On Error Resume Next
graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Item("Test")
If Err.Number > 0 Then
    graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
Else
    If MsgBox("Yes to add to existing graphics. No to start over.", vbYesNo + vbQuestion) = vbNo Then
        graphics.Delete

        graphics = partDoc.ComponentDefinition.ClientGraphicsCollection.Add("Test")
    End If
End If

' Create a new graphics node.
Dim node As GraphicsNode
node = graphics.AddNode(1)

' Do a loop that creates 5 lines that go from the minimum range point to
' five points along the maximum u parameter.
Dim startPnt As Point2d
startPnt = paramRange.MinPoint
Dim endPnt As Point2d
Dim i As Integer
Dim resultCurve As Object
Dim crvGraphics As CurveGraphics

For i = 1 To 5
    ' Create the 2d line in parametric space.
```

```
        endPnt = tg.CreatePoint2d(paramRange.MaxPoint.X, (((paramRange.MaxPoint.y - paramRange.MinPoint.y) / 4) * (i - 1)) + paramRang
        Dim surfCurve As LineSegment2d
        surfCurve = tg.CreateLineSegment2d(startPnt, endPnt)

        ' Get the equivalent 3d curve in model space.
        resultCurve = surfEval.Get3dCurveFrom2dCurve(surfCurve)

        ' Create client graphics for this curve.
        crvGraphics = node.AddCurveGraphics(resultCurve)
        crvGraphics.DepthPriority = 2
    Next

    ' Create a 2d circle in parametric space.
    Dim center As Point2d
    center = tg.CreatePoint2d((paramRange.MaxPoint.X + paramRange.MinPoint.X) / 2, (paramRange.MaxPoint.y + paramRange.MinPoint.y) / 2
    Dim radius As Double
    radius = (paramRange.MaxPoint.X - paramRange.MinPoint.X) * 0.25
    Dim paramCircle As Circle2d
    paramCircle = tg.CreateCircle2d(center, radius)

    ' Get the equivalent 3d curve in model space.
    resultCurve = surfEval.Get3dCurveFrom2dCurve(paramCircle)

    ' Create client graphics for this curve.
    crvGraphics = node.AddCurveGraphics(resultCurve)
    crvGraphics.DepthPriority = 2

    ThisApplication.ActiveView.Update
    ThisApplication.ActiveView.Fit
```

Split a 2D Curve

Description

Demonstrates the ability to extract split curves from a transient geometry curve. This sample has you select an existing sketch spline and splits it at the midpoint of parametric space. It then creates real curves using the split curve results and deletes the original sketch curve.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub SplitCurve2D()
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    Dim spline As Object
    Set spline = ThisApplication.CommandManager.Pick(kSketchCurveSplineFilter, "Select 2d spline")

    ' Get the spline geometry from the entity.
    Dim splineCurve As BSplineCurve2d
    Set splineCurve = spline.Geometry

    ' Determine the parameter value for geometric midpoint of the curve.
    Dim curveEval As Curve2dEvaluator
    Set curveEval = splineCurve.Evaluator
    Dim startParam As Double
    Dim endParam As Double
    Call curveEval.GetParamExtents(startParam, endParam)
    Dim midParam As Double
    Call curveEval.GetParamAtLength(startParam, spline.Length / 2, midParam)

    ' Split the curve.
    Dim curve1 As BSplineCurve2d
    Dim curve2 As BSplineCurve2d
    Call splineCurve.Split(midParam, curve1, curve2)

    ' Create new sketch curves using the extracted splines.
    Dim splineSketch As sketch
    Set splineSketch = spline.Parent
    Call splineSketch.SketchFixedSplines.Add(curve1)
    Call splineSketch.SketchFixedSplines.Add(curve2)

    ' Delete the original curve.
    spline.Delete
End Sub
```

To use this sample a part must be active.

Copy Code

```
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    Dim spline As Object
    spline = ThisApplication.CommandManager.Pick(kSketchCurveSplineFilter, "Select 2d spline")

    ' Get the spline geometry from the entity.
    Dim splineCurve As BSplineCurve2d
    splineCurve = spline.Geometry

    ' Determine the parameter value for geometric midpoint of the curve.
    Dim curveEval As Curve2dEvaluator
    curveEval = splineCurve.Evaluator
```

```
    Dim startParam As Double
    Dim endParam As Double
    Call curveEval.GetParamExtents(startParam, endParam)
    Dim midParam As Double
    Call curveEval.GetParamAtLength(startParam, spline.Length / 2, midParam)

    ' Split the curve.
    Dim curve1 As BSplineCurve2d
    Dim curve2 As BSplineCurve2d
    Call splineCurve.Split(midParam, curve1, curve2)

    ' Create new sketch curves using the extracted splines.
    Dim splineSketch As sketch
    splineSketch = spline.Parent
    Call splineSketch.SketchFixedSplines.Add(curve1)
    Call splineSketch.SketchFixedSplines.Add(curve2)

    ' Delete the original curve.
    spline.Delete
```

Transient solid body creation

Description

The following sample demonstrates the creation of a transient solid block body. The newly created body is then displayed using client graphics in a part.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateBlock()
    Dim oTransBRep As TransientBRep
    Set oTransBRep = ThisApplication.TransientBRep

    Dim oSurfaceBodyDef As SurfaceBodyDefinition
    Set oSurfaceBodyDef = oTransBRep.CreateSurfaceBodyDefinition

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create a lump.
    Dim oLumpDef As LumpDefinition
    Set oLumpDef = oSurfaceBodyDef.LumpDefinitions.Add

    ' Create a shell.
    Dim oShell As FaceShellDefinition
    Set oShell = oLumpDef.FaceShellDefinitions.Add

    ' Define the six planes of the box.
    Dim oPosX As Plane
    Dim oNegX As Plane
    Dim oPosY As Plane
    Dim oNegY As Plane
    Dim oPosZ As Plane
    Dim oNegZ As Plane
    Set oPosX = oTG.CreatePlane(oTG.CreatePoint(1, 0, 0), oTG.CreateVector(1, 0, 0))
    Set oNegX = oTG.CreatePlane(oTG.CreatePoint(-1, 0, 0), oTG.CreateVector(-1, 0, 0))
    Set oPosY = oTG.CreatePlane(oTG.CreatePoint(0, 1, 0), oTG.CreateVector(0, 1, 0))
    Set oNegY = oTG.CreatePlane(oTG.CreatePoint(0, -1, 0), oTG.CreateVector(0, -1, 0))
    Set oPosZ = oTG.CreatePlane(oTG.CreatePoint(0, 0, 1), oTG.CreateVector(0, 0, 1))
    Set oNegZ = oTG.CreatePlane(oTG.CreatePoint(0, 0, -1), oTG.CreateVector(0, 0, -1))

    ' Create the six faces.
    Dim oFaceDefPosX As FaceDefinition
    Dim oFaceDefNegX As FaceDefinition
    Dim oFaceDefPosY As FaceDefinition
    Dim oFaceDefNegY As FaceDefinition
    Dim oFaceDefPosZ As FaceDefinition
    Dim oFaceDefNegZ As FaceDefinition
    Set oFaceDefPosX = oShell.FaceDefinitions.Add(oPosX, False)
    Set oFaceDefNegX = oShell.FaceDefinitions.Add(oNegX, False)
    Set oFaceDefPosY = oShell.FaceDefinitions.Add(oPosY, False)
    Set oFaceDefNegY = oShell.FaceDefinitions.Add(oNegY, False)
    Set oFaceDefPosZ = oShell.FaceDefinitions.Add(oPosZ, False)
    Set oFaceDefNegZ = oShell.FaceDefinitions.Add(oNegZ, False)

    ' Create the vertices.
    Dim oVertex1 As VertexDefinition
    Dim oVertex2 As VertexDefinition
    Dim oVertex3 As VertexDefinition
    Dim oVertex4 As VertexDefinition
    Dim oVertex5 As VertexDefinition
    Dim oVertex6 As VertexDefinition
    Dim oVertex7 As VertexDefinition
    Dim oVertex8 As VertexDefinition
    Set oVertex1 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, 1, 1))
    Set oVertex2 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, 1, -1))
    Set oVertex3 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, 1, -1))
    Set oVertex4 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, 1, 1))
    Set oVertex5 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, -1, 1))
    Set oVertex6 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, -1, -1))
    Set oVertex7 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, -1, -1))
    Set oVertex8 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, -1, 1))

    ' Define the edges at intersections of the defined planes.
```

```
    Dim oEdgeDefPosXPosY As EdgeDefinition
    Dim oEdgeDefPosXNegZ As EdgeDefinition
    Dim oEdgeDefPosXNegY As EdgeDefinition
    Dim oEdgeDefPosXPosZ As EdgeDefinition
    Dim oEdgeDefNegXPosY As EdgeDefinition
    Dim oEdgeDefNegXNegZ As EdgeDefinition
    Dim oEdgeDefNegXNegY As EdgeDefinition
    Dim oEdgeDefNegXPosZ As EdgeDefinition
    Dim oEdgeDefPosYNegZ As EdgeDefinition
    Dim oEdgeDefPosYPosZ As EdgeDefinition
    Dim oEdgeDefNegYNegZ As EdgeDefinition
    Dim oEdgeDefNegYPosZ As EdgeDefinition
    Set oEdgeDefPosXPosY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex1, oVertex2, oTG.CreateLineSegment(oVertex1.Position, oVertex2.P
    Set oEdgeDefPosXNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex6, oTG.CreateLineSegment(oVertex2.Position, oVertex6.P
    Set oEdgeDefPosXNegY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex6, oVertex5, oTG.CreateLineSegment(oVertex6.Position, oVertex5.P
    Set oEdgeDefPosXPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex5, oVertex1, oTG.CreateLineSegment(oVertex5.Position, oVertex1.P
    Set oEdgeDefNegXPosY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex3, oTG.CreateLineSegment(oVertex4.Position, oVertex3.P
    Set oEdgeDefNegXNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex3, oVertex7, oTG.CreateLineSegment(oVertex3.Position, oVertex7.P
    Set oEdgeDefNegXNegY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex7, oVertex8, oTG.CreateLineSegment(oVertex7.Position, oVertex8.P
    Set oEdgeDefNegXPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex8, oVertex4, oTG.CreateLineSegment(oVertex8.Position, oVertex4.P
    Set oEdgeDefPosYNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex3, oTG.CreateLineSegment(oVertex2.Position, oVertex3.P
    Set oEdgeDefPosYPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex1, oTG.CreateLineSegment(oVertex4.Position, oVertex1.P
    Set oEdgeDefNegYNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex7, oVertex6, oTG.CreateLineSegment(oVertex7.Position, oVertex6.P
    Set oEdgeDefNegYPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex5, oVertex8, oTG.CreateLineSegment(oVertex5.Position, oVertex8.P

    ' Define the loops on the faces.
    Dim oPosXLoop As EdgeLoopDefinition
    Set oPosXLoop = oFaceDefPosX.EdgeLoopDefinitions.Add
    Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosY, True)
    Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegZ, True)
    Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegY, True)
    Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosZ, True)

    Dim oNegXLoop As EdgeLoopDefinition
    Set oNegXLoop = oFaceDefNegX.EdgeLoopDefinitions.Add
    Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosY, False)
    Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegZ, False)
    Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegY, False)
    Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosZ, False)

    Dim oPosYLoop As EdgeLoopDefinition
    Set oPosYLoop = oFaceDefPosY.EdgeLoopDefinitions.Add
    Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosY, False)
    Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosYNegZ, False)
    Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosY, True)
    Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosYPosZ, False)

    Dim oNegYLoop As EdgeLoopDefinition
    Set oNegYLoop = oFaceDefNegY.EdgeLoopDefinitions.Add
    Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegY, False)
    Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegYPosZ, False)
    Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegY, True)
    Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegYNegZ, False)

    Dim oPosZLoop As EdgeLoopDefinition
    Set oPosZLoop = oFaceDefPosZ.EdgeLoopDefinitions.Add
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosZ, True)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefNegYPosZ, True)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosZ, False)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefPosYPosZ, True)

    Dim oNegZLoop As EdgeLoopDefinition
    Set oNegZLoop = oFaceDefNegZ.EdgeLoopDefinitions.Add
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegZ, True)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefNegYNegZ, True)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegZ, False)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefPosYNegZ, True)

    ' Create a transient surface body.
    Dim oErrors As NameValueMap
    Dim oNewBody As SurfaceBody
    Set oNewBody = oSurfaceBodyDef.CreateTransientSurfaceBody(oErrors)

    ' Create client graphics to display the transient body.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oDef As PartComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

    ' Create a new graphics node within the client graphics objects.
    Dim oSurfacesNode As GraphicsNode
    Set oSurfacesNode = oClientGraphics.AddNode(1)

    ' Create client graphics based on the transient body
    Dim oSurfaceGraphics As SurfaceGraphics
    Set oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oNewBody)

    ' Update the view.
    ThisApplication.ActiveView.Update
End Sub
```

Copy Code

```
    Dim oTransBRep As TransientBRep
    oTransBRep = ThisApplication.TransientBRep

    Dim oSurfaceBodyDef As SurfaceBodyDefinition
    oSurfaceBodyDef = oTransBRep.CreateSurfaceBodyDefinition

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create a lump.
```

```
Dim oLumpDef As LumpDefinition
oLumpDef = oSurfaceBodyDef.LumpDefinitions.Add

' Create a shell.
Dim oShell As FaceShellDefinition
oShell = oLumpDef.FaceShellDefinitions.Add

' Define the six planes of the box.
Dim oPosX As Plane
Dim oNegX As Plane
Dim oPosY As Plane
Dim oNegY As Plane
Dim oPosZ As Plane
Dim oNegZ As Plane
oPosX = oTG.CreatePlane(oTG.CreatePoint(1, 0, 0), oTG.CreateVector(1, 0, 0))
oNegX = oTG.CreatePlane(oTG.CreatePoint(-1, 0, 0), oTG.CreateVector(-1, 0, 0))
oPosY = oTG.CreatePlane(oTG.CreatePoint(0, 1, 0), oTG.CreateVector(0, 1, 0))
oNegY = oTG.CreatePlane(oTG.CreatePoint(0, -1, 0), oTG.CreateVector(0, -1, 0))
oPosZ = oTG.CreatePlane(oTG.CreatePoint(0, 0, 1), oTG.CreateVector(0, 0, 1))
oNegZ = oTG.CreatePlane(oTG.CreatePoint(0, 0, -1), oTG.CreateVector(0, 0, -1))

' Create the six faces.
Dim oFaceDefPosX As FaceDefinition
Dim oFaceDefNegX As FaceDefinition
Dim oFaceDefPosY As FaceDefinition
Dim oFaceDefNegY As FaceDefinition
Dim oFaceDefPosZ As FaceDefinition
Dim oFaceDefNegZ As FaceDefinition
oFaceDefPosX = oShell.FaceDefinitions.Add(oPosX, False)
oFaceDefNegX = oShell.FaceDefinitions.Add(oNegX, False)
oFaceDefPosY = oShell.FaceDefinitions.Add(oPosY, False)
oFaceDefNegY = oShell.FaceDefinitions.Add(oNegY, False)
oFaceDefPosZ = oShell.FaceDefinitions.Add(oPosZ, False)
oFaceDefNegZ = oShell.FaceDefinitions.Add(oNegZ, False)

' Create the vertices.
Dim oVertex1 As VertexDefinition
Dim oVertex2 As VertexDefinition
Dim oVertex3 As VertexDefinition
Dim oVertex4 As VertexDefinition
Dim oVertex5 As VertexDefinition
Dim oVertex6 As VertexDefinition
Dim oVertex7 As VertexDefinition
Dim oVertex8 As VertexDefinition
oVertex1 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, 1, 1))
oVertex2 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, 1, -1))
oVertex3 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, 1, -1))
oVertex4 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, 1, 1))
oVertex5 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, -1, 1))
oVertex6 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(1, -1, -1))
oVertex7 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, -1, -1))
oVertex8 = oSurfaceBodyDef.VertexDefinitions.Add(oTG.CreatePoint(-1, -1, 1))

' Define the edges at intersections of the defined planes.
Dim oEdgeDefPosXPosY As EdgeDefinition
Dim oEdgeDefPosXNegZ As EdgeDefinition
Dim oEdgeDefPosXNegY As EdgeDefinition
Dim oEdgeDefPosXPosZ As EdgeDefinition
Dim oEdgeDefNegXPosY As EdgeDefinition
Dim oEdgeDefNegXNegZ As EdgeDefinition
Dim oEdgeDefNegXNegY As EdgeDefinition
Dim oEdgeDefNegXPosZ As EdgeDefinition
Dim oEdgeDefPosYNegZ As EdgeDefinition
Dim oEdgeDefPosYPosZ As EdgeDefinition
Dim oEdgeDefNegYNegZ As EdgeDefinition
Dim oEdgeDefNegYPosZ As EdgeDefinition
oEdgeDefPosXPosY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex1, oVertex2, oTG.CreateLineSegment(oVertex1.Position, oVertex2.Posit
oEdgeDefPosXNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex6, oTG.CreateLineSegment(oVertex2.Position, oVertex6.Posit
oEdgeDefPosXNegY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex6, oVertex5, oTG.CreateLineSegment(oVertex6.Position, oVertex5.Posit
oEdgeDefPosXPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex5, oVertex1, oTG.CreateLineSegment(oVertex5.Position, oVertex1.Posit
oEdgeDefNegXPosY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex3, oTG.CreateLineSegment(oVertex4.Position, oVertex3.Posit
oEdgeDefNegXNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex3, oVertex7, oTG.CreateLineSegment(oVertex3.Position, oVertex7.Posit
oEdgeDefNegXNegY = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex7, oVertex8, oTG.CreateLineSegment(oVertex7.Position, oVertex8.Posit
oEdgeDefNegXPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex8, oVertex4, oTG.CreateLineSegment(oVertex8.Position, oVertex4.Posit
oEdgeDefPosYNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex3, oTG.CreateLineSegment(oVertex2.Position, oVertex3.Posit
oEdgeDefPosYPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex1, oTG.CreateLineSegment(oVertex4.Position, oVertex1.Posit
oEdgeDefNegYNegZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex7, oVertex6, oTG.CreateLineSegment(oVertex7.Position, oVertex6.Posit
oEdgeDefNegYPosZ = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex5, oVertex8, oTG.CreateLineSegment(oVertex5.Position, oVertex8.Posit

' Define the loops on the faces.
Dim oPosXLoop As EdgeLoopDefinition
oPosXLoop = oFaceDefPosX.EdgeLoopDefinitions.Add
Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosY, True)
Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegZ, True)
Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegY, True)
Call oPosXLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosZ, True)

Dim oNegXLoop As EdgeLoopDefinition
oNegXLoop = oFaceDefNegX.EdgeLoopDefinitions.Add
Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosY, False)
Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegZ, False)
Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegY, False)
Call oNegXLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosZ, False)

Dim oPosYLoop As EdgeLoopDefinition
oPosYLoop = oFaceDefPosY.EdgeLoopDefinitions.Add
Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosY, False)
Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosYNegZ, False)
Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosY, True)
Call oPosYLoop.EdgeUseDefinitions.Add(oEdgeDefPosYPosZ, False)

Dim oNegYLoop As EdgeLoopDefinition
oNegYLoop = oFaceDefNegY.EdgeLoopDefinitions.Add
Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegY, False)
Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegYPosZ, False)
Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegY, True)
Call oNegYLoop.EdgeUseDefinitions.Add(oEdgeDefNegYNegZ, False)
```

```
    Dim oPosZLoop As EdgeLoopDefinition
    oPosZLoop = oFaceDefPosZ.EdgeLoopDefinitions.Add
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefNegXPosZ, True)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefNegYPosZ, True)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefPosXPosZ, False)
    Call oPosZLoop.EdgeUseDefinitions.Add(oEdgeDefPosYPosZ, True)

    Dim oNegZLoop As EdgeLoopDefinition
    oNegZLoop = oFaceDefNegZ.EdgeLoopDefinitions.Add
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefNegXNegZ, True)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefNegYNegZ, True)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefPosXNegZ, False)
    Call oNegZLoop.EdgeUseDefinitions.Add(oEdgeDefPosYNegZ, True)

    ' Create a transient surface body.
    Dim oErrors As NameValueMap
    Dim oNewBody As SurfaceBody
    oNewBody = oSurfaceBodyDef.CreateTransientSurfaceBody(oErrors)

    ' Create client graphics to display the transient body.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oDef As PartComponentDefinition
    oDef = oDoc.ComponentDefinition

    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

    ' Create a new graphics node within the client graphics objects.
    Dim oSurfacesNode As GraphicsNode
    oSurfacesNode = oClientGraphics.AddNode(1)

    ' Create client graphics based on the transient body
    Dim oSurfaceGraphics As SurfaceGraphics
    oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oNewBody)

    ' Update the view.
    ThisApplication.ActiveView.Update
```

Tapered Surface Using Offset Curve and Ruled Surface

Description

This sample demonstrates much of the wire body creation functionality. To run the sample you must have a part open and select a planar face. This sample then creates a trasient wire body using the geometry of the outside of the selected face. It then transforms and offsets that wire, and finally creates a ruled surface between the original wire and the offset wire. A base feature is created with the ruled surface.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub TaperFace()
    Dim testFace As Face
    Set testFace = ThisApplication.CommandManager.Pick(kPartFacePlanarFilter, "Select a planar face")

    ' Get the transient B-Rep and geometry objects.
    Dim tBRep As TransientBRep
    Set tBRep = ThisApplication.TransientBRep
    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Create the top-level objects needed to construct a wire.
    Dim bodyDef As SurfaceBodyDefinition
    Set bodyDef = tBRep.CreateSurfaceBodyDefinition

    Dim lumpDef As LumpDefinition
    Set lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    Set shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    Set wireDef = shellDef.WireDefinitions.Add

    ' ** Create a wire based on the outside loops of the selected face.

    ' Get the outer loop of the face.
    Dim outerLoop As EdgeLoop
    For Each outerLoop In testFace.EdgeLoops
        If outerLoop.IsOuterEdgeLoop Then
            Exit For
        End If
    Next

    ' Use the edge uses to step around the face.
    Dim lastVertexDef As VertexDefinition
    Set lastVertexDef = Nothing
    Dim coEdge As EdgeUse
    For Each coEdge In outerLoop.EdgeUses
        If lastVertexDef Is Nothing Then
            ' This is the first time through so get the starting vertex of the current edge use.
```

```
            If coEdge.IsOpposedToEdge Then
                Set lastVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StopVertex.Point)
            Else
                Set lastVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StartVertex.Point)
            End If
        End If

        ' Get the end vertex of the current edge use.
        Dim currentVertexDef As VertexDefinition
        If coEdge.IsOpposedToEdge Then
            Set currentVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StartVertex.Point)
        Else
            Set currentVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StopVertex.Point)
        End If

        ' If the geometry is an arc and the co-edge is opposed to the edge,
        ' the arc needs to be massaged so the start and end points are reversed.
        Dim geom As Object
        If coEdge.Edge.GeometryType = kCircularArcCurve And coEdge.IsOpposedToEdge Then
            Dim arc As Arc3d
            Set arc = coEdge.Edge.Geometry

            Dim center() As Double
            Dim axisVector() As Double
            Dim refVector() As Double
            Dim radius As Double
            Dim startAngle As Double
            Dim sweepAngle As Double
            Call arc.GetArcData(center, axisVector, refVector, radius, startAngle, sweepAngle)

            Set arc = tg.CreateArc3d(tg.CreatePoint(center(0), center(1), center(2)), _
                                     tg.CreateUnitVector(-axisVector(0), -axisVector(1), -axisVector(2)), _
                                     tg.CreateUnitVector(refVector(0), refVector(1), refVector(2)), _
                                     radius, startAngle + sweepAngle, sweepAngle)

            Set geom = arc
        Else
            Set geom = coEdge.Edge.Geometry
        End If


        ' Create a wire edge definition for the current edge.
        Call wireDef.WireEdgeDefinitions.Add(lastVertexDef, currentVertexDef, geom)

        ' Reset the last vertex to be the current vertex.
        Set lastVertexDef = currentVertexDef
    Next

    ' Create the wire body.
    Dim errors As NameValueMap
    Set errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim faceBody As SurfaceBody
    Set faceBody = bodyDef.CreateTransientSurfaceBody(errors)

    ' Draw the wire as client graphics to be able to visualize what's been calculated so far.
    ' This wire should overlay the selected face.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, faceBody.Wires.Item(1), True)
    MsgBox "Wire created for outer boundary of selected face."

    ' Copy the wire body and translate it along the normal of the face.
    Dim transBody As SurfaceBody
    Set transBody = tBRep.Copy(faceBody)

    Dim transMatrix As Matrix
    Set transMatrix = tg.CreateMatrix
    Dim faceNormal As Vector
    Set faceNormal = GetFaceNormal(testFace).AsVector
    Call faceNormal.ScaleBy(0.5)
    Call transMatrix.SetTranslation(faceNormal)
    Call tBRep.Transform(transBody, transMatrix)

    ' Draw the translated wire as client graphics to be able to visualize what's been calculated so far.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, transBody.Wires.Item(1), False)
    MsgBox "Copied and transformed wire created."

    ' Offset the transformed wire.
    Dim offsetWires As Wires
    Dim offsetDirection As UnitVector
    Set offsetWires = transBody.Wires.Item(1).OffsetPlanarWire(faceNormal.AsUnitVector, 0.25, kExtendCornerClosure)

    ' Draw the offset wire as client graphics to be able to visualize what's been calculated so far.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, offsetWires.Item(1), False)
    MsgBox "Offset of wire created."

    ' Creat a ruled surface between the original wire and the offset.
    Dim ruled As SurfaceBody
    Set ruled = tBRep.CreateRuledSurface(faceBody.Wires.Item(1), offsetWires.Item(1))

    ' Create a base body feature of the ruled surface.
    Dim partDef As PartComponentDefinition
    Set partDef = ThisApplication.ActiveDocument.ComponentDefinition
    Dim baseBody As NonParametricBaseFeature
    Set baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False
    MsgBox "Ruled surface between wires created and base body created."

End Sub

' Utility function that given a face returns a normal.  This is only useful
' for planar faces, since they have a consistent normal anywhere on the face.
Private Function GetFaceNormal(InputFace As Face) As UnitVector
    Dim eval As SurfaceEvaluator
    Set eval = InputFace.Evaluator

    ' Get the center of the parametric range.
```

```
    Dim center(1) As Double
    center(0) = (eval.ParamRangeRect.MinPoint.X + eval.ParamRangeRect.MaxPoint.X) / 2
    center(1) = (eval.ParamRangeRect.MinPoint.y + eval.ParamRangeRect.MaxPoint.y) / 2

    ' Calculate the normal.
    Dim normal() As Double
    Call eval.GetNormal(center, normal)

    ' Create a unit vector to pass back the result.
    Set GetFaceNormal = ThisApplication.TransientGeometry.CreateUnitVector(normal(0), normal(1), normal(2))
End Function

' Utility function that draws a Wire as client graphics.
Private Sub DrawWire(partDef As PartComponentDefinition, WireToDisplay As Wire, ClearExisting As Boolean)
    Dim trans As Transaction
    Set trans = ThisApplication.TransactionManager.StartTransaction(partDef.Document, "Wire display")

    Dim graphics As ClientGraphics
    On Error Resume Next
    Set graphics = partDef.ClientGraphicsCollection.Item("WireTest")
    If Err Then
        Set graphics = partDef.ClientGraphicsCollection.Add("WireTest")
    Else
        If ClearExisting Then
            graphics.Delete
            Set graphics = partDef.ClientGraphicsCollection.Add("WireTest")
        End If
    End If

    Dim node As GraphicsNode
    Set node = graphics.AddNode(1)

    Dim e As Edge
    For Each e In WireToDisplay.Edges
        Call node.AddCurveGraphics(e.Geometry)
    Next
    ThisApplication.ActiveView.Update

    trans.End
End Sub
```

To use this sample a part must be active.

Copy Code

```
Sub Main
    Dim testFace As Face
    testFace = ThisApplication.CommandManager.Pick(kPartFacePlanarFilter, "Select a planar face")

    ' Get the transient B-Rep and geometry objects.
    Dim tBRep As TransientBRep
    tBRep = ThisApplication.TransientBRep
    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Create the top-level objects needed to construct a wire.
    Dim bodyDef As SurfaceBodyDefinition
    bodyDef = tBRep.CreateSurfaceBodyDefinition

    Dim lumpDef As LumpDefinition
    lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    wireDef = shellDef.WireDefinitions.Add

    ' ** Create a wire based on the outside loops of the selected face.

    ' Get the outer loop of the face.
    Dim outerLoop As EdgeLoop
    For Each outerLoop In testFace.EdgeLoops
        If outerLoop.IsOuterEdgeLoop Then
            Exit For
        End If
    Next

    ' Use the edge uses to step around the face.
    Dim lastVertexDef As VertexDefinition
    lastVertexDef = Nothing
    Dim coEdge As EdgeUse
    For Each coEdge In outerLoop.EdgeUses
        If lastVertexDef Is Nothing Then
            ' This is the first time through so get the starting vertex of the current edge use.
            If coEdge.IsOpposedToEdge Then
                lastVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StopVertex.Point)
            Else
                lastVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StartVertex.Point)
            End If
        End If

        ' Get the end vertex of the current edge use.
        Dim currentVertexDef As VertexDefinition
        If coEdge.IsOpposedToEdge Then
            currentVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StartVertex.Point)
        Else
            currentVertexDef = bodyDef.VertexDefinitions.Add(coEdge.Edge.StopVertex.Point)
        End If

        ' If the geometry is an arc and the co-edge is opposed to the edge,
        ' the arc needs to be massaged so the start and end points are reversed.
        Dim geom As Object
        If coEdge.Edge.GeometryType = kCircularArcCurve And coEdge.IsOpposedToEdge Then
            Dim arc As Arc3d
            arc = coEdge.Edge.Geometry

            Dim center() As Double
```

```
            Dim axisVector() As Double
            Dim refVector() As Double
            Dim radius As Double
            Dim startAngle As Double
            Dim sweepAngle As Double
            Call arc.GetArcData(center, axisVector, refVector, radius, startAngle, sweepAngle)

            arc = tg.CreateArc3d(tg.CreatePoint(center(0), center(1), center(2)), _
                                 tg.CreateUnitVector(-axisVector(0), -axisVector(1), -axisVector(2)), _
                                 tg.CreateUnitVector(refVector(0), refVector(1), refVector(2)), _
                                 radius, startAngle + sweepAngle, sweepAngle)

            geom = arc
        Else
            geom = coEdge.Edge.Geometry
        End If


        ' Create a wire edge definition for the current edge.
        Call wireDef.WireEdgeDefinitions.Add(lastVertexDef, currentVertexDef, geom)

        ' Reset the last vertex to be the current vertex.
        lastVertexDef = currentVertexDef
    Next

    ' Create the wire body.
    Dim errors As NameValueMap
    errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim faceBody As SurfaceBody
    faceBody = bodyDef.CreateTransientSurfaceBody(errors)

    ' Draw the wire as client graphics to be able to visualize what's been calculated so far.
    ' This wire should overlay the selected face.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, faceBody.Wires.Item(1), True)
    MsgBox("Wire created for outer boundary of selected face.")

    ' Copy the wire body and translate it along the normal of the face.
    Dim transBody As SurfaceBody
    transBody = tBRep.Copy(faceBody)

    Dim transMatrix As Matrix
    transMatrix = tg.CreateMatrix
    Dim faceNormal As Vector
    faceNormal = GetFaceNormal(testFace).AsVector
    Call faceNormal.ScaleBy(0.5)
    Call transMatrix.SetTranslation(faceNormal)
    Call tBRep.Transform(transBody, transMatrix)

    ' Draw the translated wire as client graphics to be able to visualize what's been calculated so far.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, transBody.Wires.Item(1), False)
    MsgBox("Copied and transformed wire created.")

    ' Offset the transformed wire.
    Dim offsetWires As Wires
    Dim offsetDirection As UnitVector
    offsetWires = transBody.Wires.Item(1).OffsetPlanarWire(faceNormal.AsUnitVector, 0.25, kExtendCornerClosure)

    ' Draw the offset wire as client graphics to be able to visualize what's been calculated so far.
    Call DrawWire(ThisApplication.ActiveDocument.ComponentDefinition, offsetWires.Item(1), False)
    MsgBox("Offset of wire created.")

    ' Creat a ruled surface between the original wire and the offset.
    Dim ruled As SurfaceBody
    ruled = tBRep.CreateRuledSurface(faceBody.Wires.Item(1), offsetWires.Item(1))

    ' Create a base body feature of the ruled surface.
    Dim partDef As PartComponentDefinition
    partDef = ThisApplication.ActiveDocument.ComponentDefinition
    Dim baseBody As NonParametricBaseFeature
    baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False
    MsgBox("Ruled surface between wires created and base body created.")

End Sub

' Utility function that given a face returns a normal.  This is only useful
' for planar faces, since they have a consistent normal anywhere on the face.
Private Function GetFaceNormal(InputFace As Face) As UnitVector
    Dim eval As SurfaceEvaluator
    eval = InputFace.Evaluator

    ' Get the center of the parametric range.
    Dim center(1) As Double
    center(0) = (eval.ParamRangeRect.MinPoint.X + eval.ParamRangeRect.MaxPoint.X) / 2
    center(1) = (eval.ParamRangeRect.MinPoint.y + eval.ParamRangeRect.MaxPoint.y) / 2

    ' Calculate the normal.
    Dim normal() As Double = New Double() {}
    Call eval.GetNormal(center, normal)

    ' Create a unit vector to pass back the result.
    GetFaceNormal = ThisApplication.TransientGeometry.CreateUnitVector(normal(0), normal(1), normal(2))
End Function

' Utility function that draws a Wire as client graphics.
Private Sub DrawWire(partDef As PartComponentDefinition, WireToDisplay As Wire, ClearExisting As Boolean)
    Dim trans As Transaction
    trans = ThisApplication.TransactionManager.StartTransaction(partDef.Document, "Wire display")

    Dim graphics As ClientGraphics
    On Error Resume Next
    graphics = partDef.ClientGraphicsCollection.Item("WireTest")
    If Err.Number > 0 Then
        graphics = partDef.ClientGraphicsCollection.Add("WireTest")
    Else
```

```
        If ClearExisting Then
            graphics.Delete
            graphics = partDef.ClientGraphicsCollection.Add("WireTest")
        End If
    End If

    Dim node As GraphicsNode
    node = graphics.AddNode(1)

    Dim e As Edge
    For Each e In WireToDisplay.Edges
        Call node.AddCurveGraphics(e.Geometry)
    Next
    ThisApplication.ActiveView.Update

    trans.End
End Sub
```

Client graphics creation of 3D primitives

Description

This sample demonstrates the creation of 3D primitives (cylinder, cone, etc.) using client graphics.

Code Samples

- VBA
- iLogic
- C#

To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics.

Copy Code

```
Public Sub ClientGraphics3DPrimitives()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document. If it doesn't create it.
    On Error Resume Next
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Sample3DGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oClientGraphics.Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create the ClientGraphics object.
        Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oSurfacesNode As GraphicsNode
        Set oSurfacesNode = oClientGraphics.AddNode(1)

        Dim oTransientBRep As TransientBRep
        Set oTransientBRep = ThisApplication.TransientBRep

        ' Create a point representing the center of the bottom of the cone
        Dim oBottom As Point
        Set oBottom = ThisApplication.TransientGeometry.CreatePoint(0, 0, 0)

        ' Create a point representing the tip of the cone
        Dim oTop As Point
        Set oTop = ThisApplication.TransientGeometry.CreatePoint(0, 10, 0)

        ' Create a transient cone body
        Dim oBody As SurfaceBody
        Set oBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 5, 5, 0)

        ' Reset the top point indicating the center of the top of the cylinder
        Set oTop = ThisApplication.TransientGeometry.CreatePoint(0, -40, 0)

        ' Create a transient cylinder body
        Dim oCylBody As SurfaceBody
        Set oCylBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 2.5, 2.5, 2.5)

        ' Union the cone and cylinder bodies
        Call oTransientBRep.DoBoolean(oBody, oCylBody, kBooleanTypeUnion)

        ' Create client graphics based on the transient body
```

```
        Dim oSurfaceGraphics As SurfaceGraphics
        Set oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)

        ' Update the view to see the resulting curves.
        ThisApplication.ActiveView.Update
    End If
End Sub
```

To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics.

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document. If it doesn't create it.
    On Error Resume Next
    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Sample3DGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oClientGraphics.Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        oTransGeom = ThisApplication.TransientGeometry

        ' Create the ClientGraphics object.
        oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oSurfacesNode As GraphicsNode
        oSurfacesNode = oClientGraphics.AddNode(1)

        Dim oTransientBRep As TransientBRep
        oTransientBRep = ThisApplication.TransientBRep

        ' Create a point representing the center of the bottom of the cone
        Dim oBottom As Point
        oBottom = ThisApplication.TransientGeometry.CreatePoint(0, 0, 0)

        ' Create a point representing the tip of the cone
        Dim oTop As Point
        oTop = ThisApplication.TransientGeometry.CreatePoint(0, 10, 0)

        ' Create a transient cone body
        Dim oBody As SurfaceBody
        oBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 5, 5, 0)

        ' Reset the top point indicating the center of the top of the cylinder
        oTop = ThisApplication.TransientGeometry.CreatePoint(0, -40, 0)

        ' Create a transient cylinder body
        Dim oCylBody As SurfaceBody
        oCylBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 2.5, 2.5, 2.5)

        ' Union the cone and cylinder bodies
        Call oTransientBRep.DoBoolean(oBody, oCylBody, kBooleanTypeUnion)

        ' Create client graphics based on the transient body
        Dim oSurfaceGraphics As SurfaceGraphics
        oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)

        ' Update the view to see the resulting curves.
        ThisApplication.ActiveView.Update
    End If
```

To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics. The first line of this sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void ClientGraphics3DPrimitives()
{
    Application oApp = ThisApplication;
    Document oDoc = oApp.ActiveDocument;

    // Set a reference to component definition of the active document.
    // This assumes that a part or assembly document is active.
    Type t = null;
    if (oDoc.DocumentType == DocumentTypeEnum.kPartDocumentObject)
    {
        t = typeof(PartDocument);
    }
    else if (oDoc.DocumentType == DocumentTypeEnum.kAssemblyDocumentObject)
    {
        t = typeof(AssemblyDocument);
    }

    System.Reflection.PropertyInfo pi = t.GetProperty("ComponentDefinition");
    ComponentDefinition oCompDef = (ComponentDefinition)pi.GetValue(oDoc, null);
```

```
    // Check to see if the test graphics data object already exists.
    // If it does clean up by removing all associated of the client graphics
    // from the document. If it doesn't create it.
    ClientGraphics oClientGraphics;
    try
    {
        oClientGraphics = oCompDef.ClientGraphicsCollection["Sample3DGraphicsID"];

        // An existing client graphics object was successfully obtained so clean up.
        oClientGraphics.Delete();

        // update the display to see the results.
        oApp.ActiveView.Update();

    }
    catch
    {
        // Set a reference to the transient geometry object for user later.
        TransientGeometry oTransGeom = oApp.TransientGeometry;

        // Create the ClientGraphics object.
        oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Sample3DGraphicsID");

        // Create a new graphics node within the client graphics objects.
        GraphicsNode oSurfacesNode = oClientGraphics.AddNode(1);

        TransientBRep oTransientBRep = oApp.TransientBRep;

        // Create a point representing the center of the bottom of the cone
        Point oBottom = oApp.TransientGeometry.CreatePoint(0, 0, 0);

        // Create a point representing the tip of the cone
        Point oTop = oApp.TransientGeometry.CreatePoint(0, 10, 0);

        // Create a transient cone body
        SurfaceBody oBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 5, 5, 0, null);

        // Reset the top point indicating the center of the top of the cylinder
        oTop = oApp.TransientGeometry.CreatePoint(0, -40, 0);

        // Create a transient cylinder body
        SurfaceBody oCylBody = oTransientBRep.CreateSolidCylinderCone(oBottom, oTop, 2.5, 2.5, 2.5, null);

        // Union the cone and cylinder bodies
        oTransientBRep.DoBoolean(oBody, oCylBody, BooleanTypeEnum.kBooleanTypeUnion);

        // Create client graphics based on the transient body
        SurfaceGraphics oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody);

        // Update the view to see the resulting curves.
        oApp.ActiveView.Update();
    }
}
```

Transient B-Rep Ruled Surface with Lines

Description

Demonstrate creating a transient ruled surface. This sample uses all straight line segments for each of the sections. A part document must be open.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub RuledSurf()
    ' Get the transient B-Rep and Geometry objects.
    Dim tBRep As TransientBRep
    Set tBRep = ThisApplication.TransientBRep

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Create a new surface body definition.
    Dim bodyDef As SurfaceBodyDefinition
    Set bodyDef = tBRep.CreateSurfaceBodyDefinition

    ' Add a lump, shell, and wire.
    Dim lumpDef As LumpDefinition
    Set lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    Set shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    Set wireDef = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Dim pnts(2) As Point
    Set pnts(0) = tg.CreatePoint(0, 0, 0)
    Set pnts(1) = tg.CreatePoint(10, 3, 0)
    Set pnts(2) = tg.CreatePoint(20, 0, 0)

    Dim vertexDefs(2) As VertexDefinition
    Set vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
```

```
    Set vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    Set vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two wire edges, passing through the three vertices.
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a second wire definition.
    Dim wireDef2 As WireDefinition
    Set wireDef2 = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Set pnts(0) = tg.CreatePoint(-5, 0, 10)
    Set pnts(1) = tg.CreatePoint(10, 6, 10)
    Set pnts(2) = tg.CreatePoint(25, 0, 10)

    Set vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    Set vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    Set vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two edges, passing through the three vertices.
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a body using the defined wires.
    Dim errors As NameValueMap
    Set errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim body1 As SurfaceBody
    Set body1 = bodyDef.CreateTransientSurfaceBody(errors)

    ' Create a ruled surface between the two wire bodies.
    Dim ruled As SurfaceBody
    Set ruled = tBRep.CreateRuledSurface(body1.Wires.Item(1), body1.Wires.Item(2))

    ' Get the part component definition of the active document.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a base body feature of the transient body.
    Dim baseBody As NonParametricBaseFeature
    Set baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False

    ThisApplication.ActiveView.Fit
End Sub
```

To use this sample a part must be active.

Copy Code

```
    ' Get the transient B-Rep and Geometry objects.
    Dim tBRep As TransientBRep
    tBRep = ThisApplication.TransientBRep

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Create a new surface body definition.
    Dim bodyDef As SurfaceBodyDefinition
    bodyDef = tBRep.CreateSurfaceBodyDefinition

    ' Add a lump, shell, and wire.
    Dim lumpDef As LumpDefinition
    lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    wireDef = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Dim pnts(2) As Point
    pnts(0) = tg.CreatePoint(0, 0, 0)
    pnts(1) = tg.CreatePoint(10, 3, 0)
    pnts(2) = tg.CreatePoint(20, 0, 0)

    Dim vertexDefs(2) As VertexDefinition
    vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two wire edges, passing through the three vertices.
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a second wire definition.
    Dim wireDef2 As WireDefinition
    wireDef2 = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    pnts(0) = tg.CreatePoint(-5, 0, 10)
    pnts(1) = tg.CreatePoint(10, 6, 10)
    pnts(2) = tg.CreatePoint(25, 0, 10)

    vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two edges, passing through the three vertices.
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a body using the defined wires.
```

```
    Dim errors As NameValueMap
    errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim body1 As SurfaceBody
    body1 = bodyDef.CreateTransientSurfaceBody(errors)

    ' Create a ruled surface between the two wire bodies.
    Dim ruled As SurfaceBody
    ruled = tBRep.CreateRuledSurface(body1.Wires.Item(1), body1.Wires.Item(2))

    ' Get the part component definition of the active document.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a base body feature of the transient body.
    Dim baseBody As NonParametricBaseFeature
    baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False

    ThisApplication.ActiveView.Fit
```

Transient B-Rep Ruled Surface with Arc and Line

Description

Demonstrate creating a transient ruled surface. This sample uses straight line segments for once section and an arc for the second. A part document must be open.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub RuledSurf()
    ' Get the transient B-Rep and Geometry objects.
    Dim tBRep As TransientBRep
    Set tBRep = ThisApplication.TransientBRep

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Create a new surface body definition.
    Dim bodyDef As SurfaceBodyDefinition
    Set bodyDef = tBRep.CreateSurfaceBodyDefinition

    ' Add a lump, shell, and wire.
    Dim lumpDef As LumpDefinition
    Set lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    Set shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    Set wireDef = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Dim pnts(2) As Point
    Set pnts(0) = tg.CreatePoint(0, 0, 0)
    Set pnts(1) = tg.CreatePoint(10, 3, 0)
    Set pnts(2) = tg.CreatePoint(20, 0, 0)

    Dim vertexDefs(2) As VertexDefinition
    Set vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    Set vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    Set vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two wire edges, passing through the three vertices.
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a second wire definition.
    Dim wireDef2 As WireDefinition
    Set wireDef2 = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Set pnts(0) = tg.CreatePoint(-5, 0, 10)
    Set pnts(1) = tg.CreatePoint(10, 6, 10)
    Set pnts(2) = tg.CreatePoint(25, 0, 10)

    Set vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    Set vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    Set vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two edges, passing through the three vertices.
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a body using the defined wires.
    Dim errors As NameValueMap
    Set errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim body1 As SurfaceBody
    Set body1 = bodyDef.CreateTransientSurfaceBody(errors)
```

```
    ' Create a ruled surface between the two wire bodies.
    Dim ruled As SurfaceBody
    Set ruled = tBRep.CreateRuledSurface(body1.Wires.Item(1), body1.Wires.Item(2))

    ' Get the part component definition of the active document.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a base body feature of the transient body.
    Dim baseBody As NonParametricBaseFeature
    Set baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False

    ThisApplication.ActiveView.Fit
End Sub
```

To use this sample a part must be active.

Copy Code

```
    ' Get the transient B-Rep and Geometry objects.
    Dim tBRep As TransientBRep
    tBRep = ThisApplication.TransientBRep

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Create a new surface body definition.
    Dim bodyDef As SurfaceBodyDefinition
    bodyDef = tBRep.CreateSurfaceBodyDefinition

    ' Add a lump, shell, and wire.
    Dim lumpDef As LumpDefinition
    lumpDef = bodyDef.LumpDefinitions.Add

    Dim shellDef As FaceShellDefinition
    shellDef = lumpDef.FaceShellDefinitions.Add

    Dim wireDef As WireDefinition
    wireDef = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    Dim pnts(2) As Point
    pnts(0) = tg.CreatePoint(0, 0, 0)
    pnts(1) = tg.CreatePoint(10, 3, 0)
    pnts(2) = tg.CreatePoint(20, 0, 0)

    Dim vertexDefs(2) As VertexDefinition
    vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two wire edges, passing through the three vertices.
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a second wire definition.
    Dim wireDef2 As WireDefinition
    wireDef2 = shellDef.WireDefinitions.Add

    ' Create coordinate points and use those to create vertex definitions.
    pnts(0) = tg.CreatePoint(-5, 0, 10)
    pnts(1) = tg.CreatePoint(10, 6, 10)
    pnts(2) = tg.CreatePoint(25, 0, 10)

    vertexDefs(0) = bodyDef.VertexDefinitions.Add(pnts(0))
    vertexDefs(1) = bodyDef.VertexDefinitions.Add(pnts(1))
    vertexDefs(2) = bodyDef.VertexDefinitions.Add(pnts(2))

    ' Create two edges, passing through the three vertices.
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(0), vertexDefs(1), tg.CreateLineSegment(pnts(0), pnts(1)))
    Call wireDef2.WireEdgeDefinitions.Add(vertexDefs(1), vertexDefs(2), tg.CreateLineSegment(pnts(1), pnts(2)))

    ' Create a body using the defined wires.
    Dim errors As NameValueMap
    errors = ThisApplication.TransientObjects.CreateNameValueMap
    Dim body1 As SurfaceBody
    body1 = bodyDef.CreateTransientSurfaceBody(errors)

    ' Create a ruled surface between the two wire bodies.
    Dim ruled As SurfaceBody
    ruled = tBRep.CreateRuledSurface(body1.Wires.Item(1), body1.Wires.Item(2))

    ' Get the part component definition of the active document.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a base body feature of the transient body.
    Dim baseBody As NonParametricBaseFeature
    baseBody = partDef.Features.NonParametricBaseFeatures.Add(ruled)

    ' Change the result work surface so it's not translucent.
    baseBody.SurfaceBodies.Item(1).Parent.Translucent = False

    ThisApplication.ActiveView.Fit
```

Export to IFC Format Sample

Description

This sample demonstrates how to export an assembly to IFC format.

Code Samples

- VBA
- iLogic

This sample demonstrates how to export an assembly to IFC format. You should open an assembly before running below code sample.

Copy Code

```
Sub ExportToIFCFormatSample()
    ' Make sure the BIM Content addin is loaded.
    Dim oBIMContent As ApplicationAddIn
    Set oBIMContent = ThisApplication.ApplicationAddIns.ItemById("{842004D5-C360-43A8-A00D-D7EB72DAAB69}")
    If Not oBIMContent.Activated Then
        oBIMContent.Activate
    End If

    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As AssemblyComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oBIMComp As BIMComponent
    Set oBIMComp = oCompDef.BIMComponent

    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    ' specify the IFC file version to export: IFC2x3 or IFC4x3.
    oOptions.Value("IFCFileVersion") = "IFC4x3"

    ' export to IFC format with specified file version. Make sure the RCE(Revit Core Engine) is installed.
    oBIMComp.ExportBuildingComponentWithOptions "C:\Temp\MyIFC.ifc", oOptions
End Sub
```

This sample demonstrates how to export an assembly to IFC format. You should open an assembly before running below code sample.

Copy Code

```
    ' Make sure the BIM Content addin is loaded.
    Dim oBIMContent As ApplicationAddIn
    oBIMContent = ThisApplication.ApplicationAddIns.ItemById("{842004D5-C360-43A8-A00D-D7EB72DAAB69}")
    If Not oBIMContent.Activated Then
        oBIMContent.Activate
    End If

    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As AssemblyComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oBIMComp As BIMComponent
    oBIMComp = oCompDef.BIMComponent

    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    ' specify the IFC file version to export: IFC2x3 or IFC4x3.
    oOptions.Value("IFCFileVersion") = "IFC4x3"

    ' export to IFC format with specified file version. Make sure the RCE(Revit Core Engine) is installed.
    oBIMComp.ExportBuildingComponentWithOptions("C:\Temp\MyIFC.ifc", oOptions)
```

Replace content center part

Description

This sample demonstrates how to replace the content part referenced by an assembly occurrence.

Code Samples

- VBA
- iLogic

Open an assembly that contains an occurrence of a content part and run the sample.

Copy Code

```
Sub ReplaceContentCenterPart()
    ' Set a reference to the active assembly document.
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Prompt user to pick an occurrence
    Dim oOcc As ComponentOccurrence
    Set oOcc = ThisApplication.CommandManager.Pick(kAssemblyOccurrenceFilter, "Pick occurrence to replace")
```

```
    If oOcc.DefinitionDocumentType <> kPartDocumentObject Then
        MsgBox "Occurrence does not reference a content part."
        Exit Sub
    End If

    Dim oOccDef As PartComponentDefinition
    Set oOccDef = oOcc.Definition

    If Not oOccDef.IsContentMember Then
        MsgBox "The occurrence does not reference a content part."
        Exit Sub
    End If

    ' Set a reference to the ContentCenter object.
    Dim oContentCenter As ContentCenter
    Set oContentCenter = ThisApplication.ContentCenter

    ' Get the content node (category) "Fasteners:Bolts:Hex Head"
    Dim oContentNode As ContentTreeViewNode
    Set oContentNode = oContentCenter.TreeViewTopNode.ChildNodes.Item("Fasteners").ChildNodes.Item("Bolts").ChildNodes.Item("Hex Head"

    ' Get the "ISO 4015" Family object.
    Dim oFamily As ContentFamily
    For Each oFamily In oContentNode.Families
        If oFamily.DisplayName = "ISO 4015" Then
            Exit For
        End If
    Next

    ' Create a member based on the second row of the family.
    Dim error As MemberManagerErrorsEnum
    Dim strContentPartFileName As String
    Dim strErrorMessage As String
    strContentPartFileName = oFamily.CreateMember(2, error, strErrorMessage)

    Call oOcc.Replace(strContentPartFileName, False)
End Sub
```

Open an assembly that contains an occurrence of a content part and run the sample.

Copy Code

```
    ' Set a reference to the active assembly document.
    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.ActiveDocument

    ' Prompt user to pick an occurrence
    Dim oOcc As ComponentOccurrence
    oOcc = ThisApplication.CommandManager.Pick(kAssemblyOccurrenceFilter, "Pick occurrence to replace")

    If oOcc.DefinitionDocumentType <> kPartDocumentObject Then
        MsgBox("Occurrence does not reference a content part.")
        Exit Sub
    End If

    Dim oOccDef As PartComponentDefinition
    oOccDef = oOcc.Definition

    If Not oOccDef.IsContentMember Then
        MsgBox("The occurrence does not reference a content part.")
        Exit Sub
    End If

    ' Set a reference to the ContentCenter object.
    Dim oContentCenter As ContentCenter
    oContentCenter = ThisApplication.ContentCenter

    ' Get the content node (category) "Fasteners:Bolts:Hex Head"
    Dim oContentNode As ContentTreeViewNode
    oContentNode = oContentCenter.TreeViewTopNode.ChildNodes.Item("Fasteners").ChildNodes.Item("Bolts").ChildNodes.Item("Hex Head")

    ' Get the "ISO 4015" Family object.
    Dim oFamily As ContentFamily
    For Each oFamily In oContentNode.Families
        If oFamily.DisplayName = "ISO 4015" Then
            Exit For
        End If
    Next

    ' Create a member based on the second row of the family.
    Dim lError As MemberManagerErrorsEnum
    Dim strContentPartFileName As String
    Dim strErrorMessage As String
    strContentPartFileName = oFamily.CreateMember(2, lError, strErrorMessage)

    Call oOcc.Replace(strContentPartFileName, False)
```

Place Content Center Parts

Description

Places all of the items in a specified family within an assembly. The specific family is identified by the strings where it's setting the HexHeadNode variable.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PlaceFromContentCenter()
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim asmDef As AssemblyComponentDefinition
    Set asmDef = asmDoc.ComponentDefinition

    ' Get the node in the content browser based on the names of the nodes in the hierarchy.
    Dim hexHeadNode As ContentTreeViewNode
    Set hexHeadNode = ThisApplication.ContentCenter.TreeViewTopNode.ChildNodes.Item("Fasteners").ChildNodes.Item("Bolts").ChildNodes.I

    ' Find a specific family.  In this case it's using the display name, but any family
    ' characteristic could be searched for.
    Dim family As ContentFamily
    Dim checkFamily As ContentFamily
    For Each checkFamily In hexHeadNode.Families
        If checkFamily.DisplayName = "DIN EN 24016" Then
            Set family = checkFamily
            Exit For
        End If
    Next

    Dim i As Integer
    If Not family Is Nothing Then
        ' Place one instance of each member.
        Dim offset As Double
        offset = 0
        Dim row As ContentTableRow
        For Each row In family.TableRows
            ' Create the member (part file) from the table.
            Dim failureReason As MemberManagerErrorsEnum
            Dim failureMessage As String
            Dim memberFilename As String
            memberFilename = family.CreateMember(row, failureReason, failureMessage, kRefreshOutOfDateParts)

            ' Place the part into the assembly.
            Dim transMatrix As matrix
            Set transMatrix = ThisApplication.TransientGeometry.CreateMatrix
            transMatrix.Cell(2, 4) = offset
            Dim Occ As ComponentOccurrence
            Set Occ = asmDef.Occurrences.Add(memberFilename, transMatrix)

            ' Compute the position for the next placement based on the size of the part just placed.
            Dim minY As Double
            Dim maxY As Double
            minY = Occ.RangeBox.minPoint.y
            maxY = Occ.RangeBox.maxPoint.y
            offset = offset + ((maxY - minY) * 1.1)
        Next
    End If
End Sub
```

Copy Code

```
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim asmDef As AssemblyComponentDefinition
    asmDef = asmDoc.ComponentDefinition

    ' Get the node in the content browser based on the names of the nodes in the hierarchy.
    Dim hexHeadNode As ContentTreeViewNode
    hexHeadNode = ThisApplication.ContentCenter.TreeViewTopNode.ChildNodes.Item("Fasteners").ChildNodes.Item("Bolts").ChildNodes.Item(

    ' Find a specific family.  In this case it's using the display name, but any family
    ' characteristic could be searched for.
    Dim family As ContentFamily
    Dim checkFamily As ContentFamily
    For Each checkFamily In hexHeadNode.Families
        If checkFamily.DisplayName = "DIN EN 24016" Then
            family = checkFamily
            Exit For
        End If
    Next

    Dim i As Integer
    If Not family Is Nothing Then
        ' Place one instance of each member.
        Dim offset As Double
        offset = 0
        Dim row As ContentTableRow
        For Each row In family.TableRows
            ' Create the member (part file) from the table.
            Dim failureReason As MemberManagerErrorsEnum
            Dim failureMessage As String
            Dim memberFilename As String
            memberFilename = family.CreateMember(row, failureReason, failureMessage, kRefreshOutOfDateParts)

            ' Place the part into the assembly.
            Dim transMatrix As matrix
            transMatrix = ThisApplication.TransientGeometry.CreateMatrix
            transMatrix.Cell(2, 4) = offset
            Dim Occ As ComponentOccurrence
            Occ = asmDef.Occurrences.Add(memberFilename, transMatrix)

            ' Compute the position for the next placement based on the size of the part just placed.
            Dim minY As Double
            Dim maxY As Double
            minY = Occ.RangeBox.minPoint.y
            maxY = Occ.RangeBox.maxPoint.y
            offset = offset + ((maxY - minY) * 1.1)
        Next
    End If
```

Play back a simulation

Description

This sample plays back an existing dynamic simulation.

Code Samples

- VBA
- iLogic

To run this sample you must have an active document that has a valid simulation defined.

Copy Code

```
Sub PlaySimulation()
    ' Get the active document.  This assumes it is an assembly.
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.ActiveDocument

    ' The Dynamic Simulation environment must be active.
    Dim UIManager As UserInterfaceManager
    Set UIManager = ThisApplication.UserInterfaceManager
    If UIManager.ActiveEnvironment.InternalName <> "DynamicSimulationEnvironmentInternalName" Then
        ' Get the environment manager.
        Dim environmentMgr As EnvironmentManager
        Set environmentMgr = asmDoc.EnvironmentManager

        Dim dsEnv As Environment
        Set dsEnv = UIManager.Environments.Item("DynamicSimulationEnvironmentInternalName")
        Call environmentMgr.SetCurrentEnvironment(dsEnv)
    End If

    ' Get the simulation manager from the assembly.
    Dim simManager As SimulationManager
    Set simManager = asmDoc.ComponentDefinition.SimulationManager

    ' Get the first simulation.  Currently there is only ever one.
    Dim sim As DynamicSimulation
    Set sim = simManager.DynamicSimulations.Item(1)

    ' Check to see if the simulation has already been computed.
    If sim.LastComputedTimeStep < sim.NumberOfTimeSteps Then
        ' Compute the simulation, which will also play it.
        sim.ComputeSimulation
    Else
        ' Play the computed simulation.
        sim.PlaySimulation
    End If
End Sub
```

To run this sample you must have an active document that has a valid simulation defined.

Copy Code

```
    ' Get the active document.  This assumes it is an assembly.
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.ActiveDocument

    ' The Dynamic Simulation environment must be active.
    Dim UIManager As UserInterfaceManager
    UIManager = ThisApplication.UserInterfaceManager
    If UIManager.ActiveEnvironment.InternalName <> "DynamicSimulationEnvironmentInternalName" Then
        ' Get the environment manager.
        Dim environmentMgr As EnvironmentManager
        environmentMgr = asmDoc.EnvironmentManager

        Dim dsEnv As Environment
        dsEnv = UIManager.Environments.Item("DynamicSimulationEnvironmentInternalName")
        Call environmentMgr.SetCurrentEnvironment(dsEnv)
    End If

    ' Get the simulation manager from the assembly.
    Dim simManager As SimulationManager
    simManager = asmDoc.ComponentDefinition.SimulationManager

    ' Get the first simulation.  Currently there is only ever one.
    Dim sim As DynamicSimulation
    sim = simManager.DynamicSimulations.Item(1)

    ' Check to see if the simulation has already been computed.
    If sim.LastComputedTimeStep < sim.NumberOfTimeSteps Then
        ' Compute the simulation, which will also play it.
        sim.ComputeSimulation
    Else
        ' Play the computed simulation.
        sim.PlaySimulation
    End If
```

Center Dimension Text

Description

This sample demonstrates how to center the text of all dimensions on the active sheet in a drawing.

Code Samples

- VBA
- iLogic

Only linear and angular general dimensions are supported. Open a drawing document that contains several dimensions and run the sample.

Copy Code

```
Public Sub CenterAllDimensions()
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oSheet As Sheet
    Set oSheet = oDoc.ActiveSheet

    Dim oDrawingDim As DrawingDimension

    ' Iterate over all dimensions in the drawing and
    ' center them if they are linear or angular.

    For Each oDrawingDim In oSheet.DrawingDimensions
        If TypeOf oDrawingDim Is LinearGeneralDimension Or _
           TypeOf oDrawingDim Is AngularGeneralDimension Then
            Call oDrawingDim.CenterText
        End If
    Next
End Sub
```

Only linear and angular general dimensions are supported. Open a drawing document that contains several dimensions and run the sample.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oSheet As Sheet
    oSheet = oDoc.ActiveSheet

    Dim oDrawingDim As DrawingDimension

    ' Iterate over all dimensions in the drawing and
    ' center them if they are linear or angular.

    For Each oDrawingDim In oSheet.DrawingDimensions
        If TypeOf oDrawingDim Is LinearGeneralDimension Or _
           TypeOf oDrawingDim Is AngularGeneralDimension Then
            Call oDrawingDim.CenterText
        End If
    Next
```

Baseline dimension sets

Description

This sample demonstrates the creation of a baseline set dimension in a drawing.

Code Samples

- VBA
- iLogic

Create a drawing view and select multiple edges in the view before running the sample.

Copy Code

```
Public Sub CreateBaselineDimensionSet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    Dim oIntentCollection As ObjectCollection
    Set oIntentCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all the selected drawing curve segments.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Dim oDrawingCurve As DrawingCurve
    For Each oDrawingCurveSegment In oDrawDoc.SelectSet
```

```
        ' Set a reference to the drawing curve.
        Set oDrawingCurve = oDrawingCurveSegment.Parent

        Dim oDimIntent As GeometryIntent
        Set oDimIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

        Call oIntentCollection.Add(oDimIntent)
    Next

    ' Set a reference to the view to which the curve belongs.
    Dim oDrawingView As DrawingView
    Set oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the baseline dimension sets collection.
    Dim oBaselineSets As BaselineDimensionSets
    Set oBaselineSets = oActiveSheet.DrawingDimensions.BaselineDimensionSets

    ' Determine the placement point
    Dim oPlacementPoint As Point2d
    Set oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingView.Center.Y)

    ' Create a vertical baseline set dimension.
    Dim oBaselineSet As BaselineDimensionSet
    Set oBaselineSet = oBaselineSets.Add(oIntentCollection, oPlacementPoint, kVerticalDimensionType)
End Sub
```

Create a drawing view and select multiple edges in the view before running the sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    Dim oIntentCollection As ObjectCollection
    oIntentCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all the selected drawing curve segments.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Dim oDrawingCurve As DrawingCurve
    For Each oDrawingCurveSegment In oDrawDoc.SelectSet

        ' Set a reference to the drawing curve.
        oDrawingCurve = oDrawingCurveSegment.Parent

        Dim oDimIntent As GeometryIntent
        oDimIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

        Call oIntentCollection.Add(oDimIntent)
    Next

    ' Set a reference to the view to which the curve belongs.
    Dim oDrawingView As DrawingView
    oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the baseline dimension sets collection.
    Dim oBaselineSets As BaselineDimensionSets
    oBaselineSets = oActiveSheet.DrawingDimensions.BaselineDimensionSets

    ' Determine the placement point
    Dim oPlacementPoint As Point2d
    oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingView.Center.Y)

    ' Create a vertical baseline set dimension.
    Dim oBaselineSet As BaselineDimensionSet
    oBaselineSet = oBaselineSets.Add(oIntentCollection, oPlacementPoint, kVerticalDimensionType)
```

Create bend note

Description

This sample demonstrates the creation of a bend note on the drawing view of a flat pattern.

Code Samples

- VBA
- iLogic

Select a bend edge on a flat pattern drawing view and run the sample.

Copy Code

```
Sub AddBendNote()
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a bend edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox "A single bend edge must be selected."
        Exit Sub
    End If
```

```
    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox "A bend edge must be selected."
        Exit Sub
    End If

    Dim oBendEdge As DrawingCurve
    Set oBendEdge = oDoc.SelectSet(1).Parent

    If Not (oBendEdge.EdgeType = kBendUpEdge Or oBendEdge.EdgeType = kBendDownEdge) Then
        MsgBox "A bend edge must be selected."
        Exit Sub
    End If

    ' Create the bend note
    Dim oBendNote As BendNote
    Set oBendNote = oDoc.ActiveSheet.DrawingNotes.BendNotes.Add(oBendEdge)
End Sub
```

Select a bend edge on a flat pattern drawing view and run the sample.

Copy Code

```
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a bend edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox("A single bend edge must be selected.")
        Exit Sub
    End If

    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox("A bend edge must be selected.")
        Exit Sub
    End If

    Dim oBendEdge As DrawingCurve
    oBendEdge = oDoc.SelectSet(1).Parent

    If Not (oBendEdge.EdgeType = kBendUpEdge Or oBendEdge.EdgeType = kBendDownEdge) Then
        MsgBox("A bend edge must be selected.")
        Exit Sub
    End If

    ' Create the bend note
    Dim oBendNote As BendNote
    oBendNote = oDoc.ActiveSheet.DrawingNotes.BendNotes.Add(oBendEdge)
```

Chain dimensions sets

Description

This sample demonstrates the creation of a chain dimension set in a drawing.

Code Samples

- VBA
- iLogic

Create a drawing view and select multiple edges in the view before running the sample.

Copy Code

```
Public Sub CreateChainDimensionSet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    Dim oIntentCollection As ObjectCollection
    Set oIntentCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all the selected drawing curve segments.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Dim oDrawingCurve As DrawingCurve
    For Each oDrawingCurveSegment In oDrawDoc.SelectSet

        ' Set a reference to the drawing curve.
        Set oDrawingCurve = oDrawingCurveSegment.Parent

        Dim oDimIntent As GeometryIntent
        Set oDimIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

        Call oIntentCollection.Add(oDimIntent)
    Next

   ' Set a reference to the view to which the curve belongs.
    Dim oDrawingView As DrawingView
    Set oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the chain dimension sets collection.
    Dim oChainDimSets As ChainDimensionSets
    Set oChainDimSets = oActiveSheet.DrawingDimensions.ChainDimensionSets
```

```
    ' Determine the placement point
    Dim oPlacementPoint As Point2d
    Set oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Center.X, oDrawingView.Top + 5)

    ' Create a horizontal chain dimension set.
    Dim oChainDimSet As ChainDimensionSet
    Set oChainDimSet = oChainDimSets.Add(oIntentCollection, oPlacementPoint, kHorizontalDimensionType)
End Sub
```

Create a drawing view and select multiple edges in the view before running the sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    Dim oIntentCollection As ObjectCollection
    oIntentCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all the selected drawing curve segments.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Dim oDrawingCurve As DrawingCurve
    For Each oDrawingCurveSegment In oDrawDoc.SelectSet

        ' Set a reference to the drawing curve.
        oDrawingCurve = oDrawingCurveSegment.Parent

        Dim oDimIntent As GeometryIntent
        oDimIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

        Call oIntentCollection.Add(oDimIntent)
    Next

   ' Set a reference to the view to which the curve belongs.
    Dim oDrawingView As DrawingView
    oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the chain dimension sets collection.
    Dim oChainDimSets As ChainDimensionSets
    oChainDimSets = oActiveSheet.DrawingDimensions.ChainDimensionSets

    ' Determine the placement point
    Dim oPlacementPoint As Point2d
    oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Center.X, oDrawingView.Top + 5)

    ' Create a horizontal chain dimension set.
    Dim oChainDimSet As ChainDimensionSet
    oChainDimSet = oChainDimSets.Add(oIntentCollection, oPlacementPoint, kHorizontalDimensionType)
```

Bullet and Numbering List

Description

This sample demonstrates how to create bullets and numbering list in a drawing note.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub DrawingNoteBulletAndNumberingListSample()
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)


    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(12, 12)

    Dim sFormattedText As String
    sFormattedText = "<Bullet>Line1</Bullet><Bullet>Line2</Bullet><Bullet>Line3</Bullet><Numbering Format='a'>Number1</Numbering><Numb

    Dim oNote As DrawingNote
    Set oNote = oDoc.Sheets(1).DrawingNotes.GeneralNotes.AddFitted(oPosition, sFormattedText)

    Debug.Print oNote.Text
    Debug.Print oNote.FormattedText
End Sub
```

Copy Code

```
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)


    Dim oPosition As Point2d
    oPosition = ThisApplication.TransientGeometry.CreatePoint2d(12, 12)

    Dim sFormattedText As String
    sFormattedText = "<Bullet>Line1</Bullet><Bullet>Line2</Bullet><Bullet>Line3</Bullet><Numbering Format='a'>Number1</Numbering><Numb
```

```
    Dim oNote As DrawingNote
    oNote = oDoc.Sheets(1).DrawingNotes.GeneralNotes.AddFitted(oPosition, sFormattedText)

    Logger.Info(oNote.Text)

    Logger.Info(oNote.FormattedText)
```

Drawing Welding Symbol Creation

Description

This sample is to demonstrate how to create a drawing welding symbol.

Code Samples

- VBA
- iLogic

This sample is to demonstrate how to create a drawing welding symbol.

Copy Code

```
Public Sub AddDrawingWeldingSymbolSample()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' Please select a linear drwaing curve.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Set oDrawingCurveSegment = ThisApplication.CommandManager.Pick(kDrawingCurveSegmentFilter, "Select a linear curve")

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    Set oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    Set oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)

    Dim oWeldingSymDefs As DrawingWeldingSymbolDefinitions
    Set oWeldingSymDefs = oActiveSheet.WeldingSymbols.CreateDefinitions()

    Dim oWeldingSymDef As DrawingWeldingSymbolDefinition
    Set oWeldingSymDef = oWeldingSymDefs.Add(1)

    ' Specify the weld symbol type(WeldSymbolTypeEnum/BackingSymbolTypeEnum)
    oWeldingSymDef.WeldSymbolOne.WeldSymbolType = BackingSymbolTypeEnum.kConsumableInsertANSI
    oWeldingSymDef.WeldSymbolTwo.WeldSymbolType = WeldSymbolTypeEnum.kNoneWeldSymbolType
    oWeldingSymDef.ClosedNoteTail = True
    oWeldingSymDef.FieldWeldingSymbol = True

    ' Create the symbol with a leader
    Dim oSymbol As DrawingWeldingSymbol
    Set oSymbol = oActiveSheet.WeldingSymbols.Add(oLeaderPoints, oWeldingSymDefs)
End Sub
```

This sample is to demonstrate how to create a drawing welding symbol.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' Please select a linear drwaing curve.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    oDrawingCurveSegment = ThisApplication.CommandManager.Pick(kDrawingCurveSegmentFilter, "Select a linear curve")
```

```
    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)

    Dim oWeldingSymDefs As DrawingWeldingSymbolDefinitions
    oWeldingSymDefs = oActiveSheet.WeldingSymbols.CreateDefinitions()

    Dim oWeldingSymDef As DrawingWeldingSymbolDefinition
    oWeldingSymDef = oWeldingSymDefs.Add(1)

    ' Specify the weld symbol type(WeldSymbolTypeEnum/BackingSymbolTypeEnum)
    oWeldingSymDef.WeldSymbolOne.WeldSymbolType = BackingSymbolTypeEnum.kConsumableInsertANSI
    oWeldingSymDef.WeldSymbolTwo.WeldSymbolType = WeldSymbolTypeEnum.kNoneWeldSymbolType
    oWeldingSymDef.ClosedNoteTail = True
    oWeldingSymDef.FieldWeldingSymbol = True

    ' Create the symbol with a leader
    Dim oSymbol As DrawingWeldingSymbol
    oSymbol = oActiveSheet.WeldingSymbols.Add(oLeaderPoints, oWeldingSymDefs)
```

EdgeSymbol Creation Sample

Description

This sample is to demonstrate how to create a EdgeSymbol in drawing document.

Code Samples

- VBA
- iLogic

This sample is to demonstrate how to create a EdgeSymbol in drawing document.

Copy Code

```
Public Sub CreateEdgeSymbol()

    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Specify the leader points
    Dim oPoints(1 To 2) As Point2d
    Set oPoints(1) = oTG.CreatePoint2d(10, 10)
    Set oPoints(2) = oTG.CreatePoint2d(13, 15)

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 1 To 2
        Call oLeaderPoints.Add(oPoints(i))
    Next

    Dim oEdgeSymbolDef As EdgeSymbolDefinition
    Set oEdgeSymbolDef = oActiveSheet.EdgeSymbols.CreateDefinition(kEdgeSymbolValueNoValues, kAllEdgesIndicationType)

    ' Create teh edge symbol.
    Dim oEdgeSymbol As EdgeSymbol
    Set oEdgeSymbol = oActiveSheet.EdgeSymbols.Add(oLeaderPoints, oEdgeSymbolDef)
End Sub
```

This sample is to demonstrate how to create a EdgeSymbol in drawing document.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Specify the leader points
    Dim oPoints(0 To 1) As Point2d
    oPoints(0) = oTG.CreatePoint2d(10, 10)
    oPoints(1) = oTG.CreatePoint2d(13, 15)

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 0 To 1
        Call oLeaderPoints.Add(oPoints(i))
    Next

    Dim oEdgeSymbolDef As EdgeSymbolDefinition
    oEdgeSymbolDef = oActiveSheet.EdgeSymbols.CreateDefinition(kEdgeSymbolValueNoValues, kAllEdgesIndicationType)

    ' Create teh edge symbol.
    Dim oEdgeSymbol As EdgeSymbol
    oEdgeSymbol = oActiveSheet.EdgeSymbols.Add(oLeaderPoints, oEdgeSymbolDef)
```

Create a feature control frame symbol

Description

This sample demonstrates the creation of a feature control frame symbol.

Code Samples

- VBA
- iLogic

Select a linear drawing curve and run the sample.

Copy Code

```
Public Sub AddFeatureControlFrame()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Set oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    Set oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    Set oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)
    Call oLeaderPoints.Add(oGeometryIntent)

    ' Create a FeatureControlFrameRows object to define the symbol's rows
    Dim oRows As FeatureControlFrameRows
    Set oRows = oActiveSheet.FeatureControlFrames.CreateFeatureControlFrameRows

    ' Add a row
    Dim oRow As FeatureControlFrameRow
    Set oRow = oRows.Add(kProfileOfAnySurface, "0.20", , "A", "B")

    ' Create the feature control frame symbol with a leader
    Dim oSymbol As FeatureControlFrame
```

```
    Set oSymbol = oActiveSheet.FeatureControlFrames.Add(oLeaderPoints, oRows)
End Sub
```

Select a linear drawing curve and run the sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)
    Call oLeaderPoints.Add(oGeometryIntent)

    ' Create a FeatureControlFrameRows object to define the symbol's rows
    Dim oRows As FeatureControlFrameRows
    oRows = oActiveSheet.FeatureControlFrames.CreateFeatureControlFrameRows

    ' Add a row
    Dim oRow As FeatureControlFrameRow
    oRow = oRows.Add(kProfileOfAnySurface, "0.20", , "A", "B")

    ' Create the feature control frame symbol with a leader
    Dim oSymbol As FeatureControlFrame
    oSymbol = oActiveSheet.FeatureControlFrames.Add(oLeaderPoints, oRows)
```

Add a general note

Description

This sample illustrates creating text (general note) in a sheet.

Code Samples

- VBA
- iLogic

Before running this sample, open a drawing document.

Copy Code

```
Public Sub SheetTextAdd()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the GeneralNotes object
    Dim oGeneralNotes As GeneralNotes
    Set oGeneralNotes = oActiveSheet.DrawingNotes.GeneralNotes

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create text with simple string as input. Since this doesn't use
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "Drawing Notes"

    Dim oGeneralNote As GeneralNote
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, 18), sText)
```

```
    ' Create text using various overrides.
    sText = "Notice: All holes larger than 0.500 n are to be lubricated."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, 16), sText)

    ' Create a set of notes that are numbered and aligned along the left.
    Dim dYCoord As Double
    dYCoord = 14
    Dim dYOffset As Double
    Dim oStyle As TextStyle
    Set oStyle = oGeneralNotes.Item(1).TextStyle
    dYOffset = oStyle.FontSize * 1.5

    ' Simple single line text.
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "1.")
    sText = "This is note 1."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Two line text. The two lines are defined using the  tag within the text string.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "2.")
    sText = "This is note 2,  which contains two lines."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Single line of text.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "3.")
    sText = "This is note 3."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Three lines of text.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "4.")
    sText = "This is note 4,  which contains  several lines."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "5.")

    sText = "Here is the last and final line of text."
    Set oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)
End Sub
```

Before running this sample, open a drawing document.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the GeneralNotes object
    Dim oGeneralNotes As GeneralNotes
    oGeneralNotes = oActiveSheet.DrawingNotes.GeneralNotes

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create text with simple string as input. Since this doesn't use
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "Drawing Notes"

    Dim oGeneralNote As GeneralNote
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, 18), sText)

    ' Create text using various overrides.
    sText = "Notice: All holes larger than 0.500 n are to be lubricated."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, 16), sText)

    ' Create a set of notes that are numbered and aligned along the left.
    Dim dYCoord As Double
    dYCoord = 14
    Dim dYOffset As Double
    Dim oStyle As TextStyle
    oStyle = oGeneralNotes.Item(1).TextStyle
    dYOffset = oStyle.FontSize * 1.5

    ' Simple single line text.
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "1.")
    sText = "This is note 1."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Two line text. The two lines are defined using the  tag within the text string.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "2.")
    sText = "This is note 2,  which contains two lines."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Single line of text.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "3.")
    sText = "This is note 3."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Three lines of text.
    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "4.")
    sText = "This is note 4,  which contains  several lines."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    dYCoord = dYCoord - (oGeneralNote.FittedTextHeight + 0.5)
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "5.")
```

```
    sText = "Here is the last and final line of text."
    oGeneralNote = oGeneralNotes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)
```

Creating Stacked Text

Description

This sample demonstrates the creation of stacked text and text with superscript & subscript.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub StackedText()
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the text
    Dim oPoint As Point2d
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Define horizontally stacked text
    Dim strHorizontalStack As String
    strHorizontalStack = "11/2"

    ' Define diagonally stacked text
    Dim strDiagonalStack As String
    strDiagonalStack = "11#2"

    ' Define text with subscript
    Dim strSubscript As String
    strSubscript = "H^2SO^4"

    ' Define text with superscript
    Dim strSuperscript As String
    strSuperscript = "x2^ + y2^ = z2^"

    Dim strText As String
    strText = strHorizontalStack & "" & strDiagonalStack & "" & strSubscript & "" & strSuperscript

    Dim oGeneralNote As GeneralNote
    Set oGeneralNote = oActiveSheet.DrawingNotes.GeneralNotes.AddFitted(oPoint, strText)
End Sub
```

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the text
    Dim oPoint As Point2d
    oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Define horizontally stacked text
    Dim strHorizontalStack As String
    strHorizontalStack = "11/2"

    ' Define diagonally stacked text
    Dim strDiagonalStack As String
    strDiagonalStack = "11#2"

    ' Define text with subscript
    Dim strSubscript As String
    strSubscript = "H^2SO^4"

    ' Define text with superscript
    Dim strSuperscript As String
    strSuperscript = "x2^ + y2^ = z2^"

    Dim strText As String
    strText = strHorizontalStack & "" & strDiagonalStack & "" & strSubscript & "" & strSuperscript

    Dim oGeneralNote As GeneralNote
    oGeneralNote = oActiveSheet.DrawingNotes.GeneralNotes.AddFitted(oPoint, strText)
```

Create thread note

Description

This sample demonstrates the creation of a thread note on a drawing view.

Code Samples

- VBA
- iLogic

Select a thread edge on a drawing view and run the sample.

Copy Code

```
Sub AddThreadNote()
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a thread edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox "A single thread edge must be selected."
        Exit Sub
    End If

    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox "A thread edge must be selected."
        Exit Sub
    End If

    Dim oThreadEdge As DrawingCurve
    Set oThreadEdge = oDoc.SelectSet(1).Parent

    If Not (oThreadEdge.EdgeType = kThreadEdge) Then
        MsgBox "A thread edge must be selected."
        Exit Sub
    End If

    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(5, 25)

    ' Create the thread note
    Dim oThreadNote As HoleThreadNote
    Set oThreadNote = oDoc.ActiveSheet.DrawingNotes.HoleThreadNotes.Add(oPosition, oThreadEdge)
End Sub
```

Select a thread edge on a drawing view and run the sample.

Copy Code

```
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a thread edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox("A single thread edge must be selected.")
        Exit Sub
    End If

    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox("A thread edge must be selected.")
        Exit Sub
    End If

    Dim oThreadEdge As DrawingCurve
    oThreadEdge = oDoc.SelectSet(1).Parent

    If Not (oThreadEdge.EdgeType = kThreadEdge) Then
        MsgBox("A thread edge must be selected.")
        Exit Sub
    End If

    Dim oPosition As Point2d
    oPosition = ThisApplication.TransientGeometry.CreatePoint2d(5, 25)

    ' Create the thread note
    Dim oThreadNote As HoleThreadNote
    oThreadNote = oDoc.ActiveSheet.DrawingNotes.HoleThreadNotes.Add(oPosition, oThreadEdge)
```

Add new leader note

Description

This sample illustrates creating leader text on a sheet.

Code Samples

- VBA
- iLogic

Before running this sample, open a drawing document and select a linear drawing edge.

Copy Code

```
Public Sub AddLeaderNote()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drawing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Set oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    Set oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selected curve is linear
    Dim oMidPoint As Point2d
    Set oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the leader text will attach to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)

    ' Create text with simple string as input. Since this doesn't use
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "API Leader Note"

    Dim oLeaderNote As LeaderNote
    Set oLeaderNote = oActiveSheet.DrawingNotes.LeaderNotes.Add(oLeaderPoints, sText)

    ' Insert a node.
    Dim oFirstNode As LeaderNode
    Set oFirstNode = oLeaderNote.Leader.RootNode.ChildNodes.Item(1)

    Dim oSecondNode As LeaderNode
    Set oSecondNode = oFirstNode.ChildNodes.Item(1)

    Call oFirstNode.InsertNode(oSecondNode, oTG.CreatePoint2d(oMidPoint.X + 5, oMidPoint.Y + 5))
End Sub
```

Before running this sample, open a drawing document and select a linear drawing edge.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drawing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selected curve is linear
    Dim oMidPoint As Point2d
    oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the leader text will attach to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)
```

```
    ' Create text with simple string as input. Since this doesn't use
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "API Leader Note"

    Dim oLeaderNote As LeaderNote
    oLeaderNote = oActiveSheet.DrawingNotes.LeaderNotes.Add(oLeaderPoints, sText)

    ' Insert a node.
    Dim oFirstNode As LeaderNode
    oFirstNode = oLeaderNote.Leader.RootNode.ChildNodes.Item(1)

    Dim oSecondNode As LeaderNode
    oSecondNode = oFirstNode.ChildNodes.Item(1)

    Call oFirstNode.InsertNode(oSecondNode, oTG.CreatePoint2d(oMidPoint.X + 5, oMidPoint.Y + 5))
```

Create ordinate dimension

Description

This sample demonstrates the creation of ordinate dimensions in a drawing document.

Code Samples

- VBA
- iLogic

Before running this sample, select a linear drawing curve on the active drawing document.

Copy Code

```
Public Sub CreateOrdinateDimensions()
  ' Set a reference to the drawing document.
  ' This assumes a drawing document is active.
  Dim oDrawDoc As DrawingDocument
  Set oDrawDoc = ThisApplication.ActiveDocument

  ' Set a reference to the active sheet.
  Dim oActiveSheet As Sheet
  Set oActiveSheet = oDrawDoc.ActiveSheet

  ' Set a reference to the drawing curve segment.
  ' This assumes that a linear drawing curve is selected.
  Dim oDrawingCurveSegment As DrawingCurveSegment
  Set oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

  ' Set a reference to the drawing curve.
  Dim oDrawingCurve As DrawingCurve
  Set oDrawingCurve = oDrawingCurveSegment.Parent

  If Not oDrawingCurve.CurveType = kLineSegmentCurve Then
    MsgBox "A linear curve should be selected for this sample."
    Exit Sub
  End If

  ' Create point intents to anchor the dimension to.
  Dim oDimIntent1 As GeometryIntent
  Set oDimIntent1 = oActiveSheet.CreateGeometryIntent(oDrawingCurve, kStartPointIntent)

  Dim oDimIntent2 As GeometryIntent
  Set oDimIntent2 = oActiveSheet.CreateGeometryIntent(oDrawingCurve, kEndPointIntent)

  ' Set a reference to the view to which the curve belongs.
  Dim oDrawingView As DrawingView
  Set oDrawingView = oDrawingCurve.Parent

  ' If origin indicator has not been already created, create it first.
  If Not oDrawingView.HasOriginIndicator Then
    ' The indicator will be located at the start point of the selected curve.
    oDrawingView.CreateOriginIndicator oDimIntent1
  End If

  ' Set a reference to the ordinate dimensions collection.
  Dim oOrdinateDimensions As OrdinateDimensions
  Set oOrdinateDimensions = oActiveSheet.DrawingDimensions.OrdinateDimensions

  ' Create the x-axis vector
  Dim oXAxis As Vector2d
  Set oXAxis = ThisApplication.TransientGeometry.CreateVector2d(1, 0)

  Dim oCurveVector As Vector2d
  Set oCurveVector = oDrawingCurve.StartPoint.VectorTo(oDrawingCurve.EndPoint)

  Dim oTextOrigin1 As Point2d
  Dim oTextOrigin2 As Point2d
  Dim DimType As DimensionTypeEnum

  If oCurveVector.IsParallelTo(oXAxis) Then
    ' Selected curve is horizontal
    DimType = kVerticalDimensionType

    ' Set the text points for the 2 dimensions.
    Set oTextOrigin1 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingCurve.StartPoint.X, oDrawingView.Top + 5)
    Set oTextOrigin2 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingCurve.EndPoint.X, oDrawingView.Top + 5)

  Else
    ' Selected curve is vertical or at an angle.
    DimType = kHorizontalDimensionType
```

```
    ' Set the text points for the 2 dimensions.
    Set oTextOrigin1 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingCurve.StartPoint.Y)
    Set oTextOrigin2 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingCurve.EndPoint.Y)

  End If

  ' Create the first ordinate dimension.
  Dim oOrdinateDimension1 As OrdinateDimension
  Set oOrdinateDimension1 = oOrdinateDimensions.Add(oDimIntent1, oTextOrigin1, DimType)

  ' Create the second ordinate dimension.
  Dim oOrdinateDimension2 As OrdinateDimension
  Set oOrdinateDimension2 = oOrdinateDimensions.Add(oDimIntent2, oTextOrigin2, DimType)
End Sub
```

Before running this sample, select a linear drawing curve on the active drawing document.

Copy Code

```
    ' Set a reference to the drawing document.
  ' This assumes a drawing document is active.
  Dim oDrawDoc As DrawingDocument
  oDrawDoc = ThisApplication.ActiveDocument

  ' Set a reference to the active sheet.
  Dim oActiveSheet As Sheet
  oActiveSheet = oDrawDoc.ActiveSheet

  ' Set a reference to the drawing curve segment.
  ' This assumes that a linear drawing curve is selected.
  Dim oDrawingCurveSegment As DrawingCurveSegment
  oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

  ' Set a reference to the drawing curve.
  Dim oDrawingCurve As DrawingCurve
  oDrawingCurve = oDrawingCurveSegment.Parent

  If Not oDrawingCurve.CurveType = kLineSegmentCurve Then
    MsgBox("A linear curve should be selected for this sample.")
    Exit Sub
  End If

  ' Create point intents to anchor the dimension to.
  Dim oDimIntent1 As GeometryIntent
  oDimIntent1 = oActiveSheet.CreateGeometryIntent(oDrawingCurve, kStartPointIntent)

  Dim oDimIntent2 As GeometryIntent
  oDimIntent2 = oActiveSheet.CreateGeometryIntent(oDrawingCurve, kEndPointIntent)

  ' Set a reference to the view to which the curve belongs.
  Dim oDrawingView As DrawingView
  oDrawingView = oDrawingCurve.Parent

  ' If origin indicator has not been already created, create it first.
  If Not oDrawingView.HasOriginIndicator Then
    ' The indicator will be located at the start point of the selected curve.
    oDrawingView.CreateOriginIndicator(oDimIntent1)
  End If

  ' Set a reference to the ordinate dimensions collection.
  Dim oOrdinateDimensions As OrdinateDimensions
  oOrdinateDimensions = oActiveSheet.DrawingDimensions.OrdinateDimensions

  ' Create the x-axis vector
  Dim oXAxis As Vector2d
  oXAxis = ThisApplication.TransientGeometry.CreateVector2d(1, 0)

  Dim oCurveVector As Vector2d
  oCurveVector = oDrawingCurve.StartPoint.VectorTo(oDrawingCurve.EndPoint)

  Dim oTextOrigin1 As Point2d
  Dim oTextOrigin2 As Point2d
  Dim DimType As DimensionTypeEnum

  If oCurveVector.IsParallelTo(oXAxis) Then
    ' Selected curve is horizontal
    DimType = kVerticalDimensionType

    ' Set the text points for the 2 dimensions.
    oTextOrigin1 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingCurve.StartPoint.X, oDrawingView.Top + 5)
    oTextOrigin2 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingCurve.EndPoint.X, oDrawingView.Top + 5)

  Else
    ' Selected curve is vertical or at an angle.
    DimType = kHorizontalDimensionType

    ' Set the text points for the 2 dimensions.
    oTextOrigin1 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingCurve.StartPoint.Y)
    oTextOrigin2 = ThisApplication.TransientGeometry.CreatePoint2d(oDrawingView.Left - 5, oDrawingCurve.EndPoint.Y)

  End If

  ' Create the first ordinate dimension.
  Dim oOrdinateDimension1 As OrdinateDimension
  oOrdinateDimension1 = oOrdinateDimensions.Add(oDimIntent1, oTextOrigin1, DimType)

  ' Create the second ordinate dimension.
  Dim oOrdinateDimension2 As OrdinateDimension
  oOrdinateDimension2 = oOrdinateDimensions.Add(oDimIntent2, oTextOrigin2, DimType)
```

create punch note

Description

This sample demonstrates the creation of a punch note on the drawing view of a flat pattern.

Code Samples

- VBA
- iLogic

Select a punch edge on a flat pattern drawing view and run the sample.

Copy Code

```
Sub AddPunchNote()
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a punch edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox "A single punch edge must be selected."
        Exit Sub
    End If

    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox "A punch edge must be selected."
        Exit Sub
    End If

    Dim oPunchEdge As DrawingCurve
    Set oPunchEdge = oDoc.SelectSet(1).Parent

    If Not (oPunchEdge.EdgeType = kPunchUpEdge Or oPunchEdge.EdgeType = kPunchDownEdge) Then
        MsgBox "A punch edge must be selected."
        Exit Sub
    End If

    Dim oPunchEdgeIntent As GeometryIntent
    Set oPunchEdgeIntent = oDoc.ActiveSheet.CreateGeometryIntent(oPunchEdge)

    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(5, 25)

    ' Create the punch note
    Dim oPunchNote As PunchNote
    Set oPunchNote = oDoc.ActiveSheet.DrawingNotes.PunchNotes.Add(oPosition, oPunchEdgeIntent)
End Sub
```

Select a punch edge on a flat pattern drawing view and run the sample.

Copy Code

```
    ' Assumes that a drawing document is active
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a punch edge is selected.
    If oDoc.SelectSet.Count <> 1 Then
        MsgBox("A single punch edge must be selected.")
        Exit Sub
    End If

    If Not TypeOf oDoc.SelectSet(1) Is DrawingCurveSegment Then
        MsgBox("A punch edge must be selected.")
        Exit Sub
    End If

    Dim oPunchEdge As DrawingCurve
    oPunchEdge = oDoc.SelectSet(1).Parent

    If Not (oPunchEdge.EdgeType = kPunchUpEdge Or oPunchEdge.EdgeType = kPunchDownEdge) Then
        MsgBox("A punch edge must be selected.")
        Exit Sub
    End If

    Dim oPunchEdgeIntent As GeometryIntent
    oPunchEdgeIntent = oDoc.ActiveSheet.CreateGeometryIntent(oPunchEdge)

    Dim oPosition As Point2d
    oPosition = ThisApplication.TransientGeometry.CreatePoint2d(5, 25)

    ' Create the punch note
    Dim oPunchNote As PunchNote
    oPunchNote = oDoc.ActiveSheet.DrawingNotes.PunchNotes.Add(oPosition, oPunchEdgeIntent)
```

RevisionCloud Creation Sample

Description

This sample is to demonstrate how to create a revision cloud in drawing document.

Code Samples

- VBA
- iLogic

This sample is to demonstrate how to create a revision cloud in drawing document.

Copy Code

```
Public Sub CreateRevisionCloud()

    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Specify the control points position
    Dim oPosition(1 To 4) As Point2d
    Set oPosition(1) = oTG.CreatePoint2d(10, 10)
    Set oPosition(2) = oTG.CreatePoint2d(13, 15)
    Set oPosition(3) = oTG.CreatePoint2d(18, 10)
    Set oPosition(4) = oTG.CreatePoint2d(13, 6)

    Dim oControlPoints As ObjectCollection
    Set oControlPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 1 To 4
        Call oControlPoints.Add(oPosition(i))
    Next

    ' Create teh revision cloud definition object.
    Dim oRCDef As RevisionCloudDefinition
    Set oRCDef = oActiveSheet.RevisionClouds.CreateRevisionCloudDefinition(oControlPoints)

    ' Create the revision cloud.
    Dim oRevisionCloud As RevisionCloud
    Set oRevisionCloud = oActiveSheet.RevisionClouds.Add(oRCDef)

End Sub
```

This sample is to demonstrate how to create a revision cloud in drawing document.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Specify the control points position
    Dim oPosition(0 To 3) As Point2d
    oPosition(0) = oTG.CreatePoint2d(10, 10)
    oPosition(1) = oTG.CreatePoint2d(13, 15)
    oPosition(2) = oTG.CreatePoint2d(18, 10)
    oPosition(3) = oTG.CreatePoint2d(13, 6)

    Dim oControlPoints As ObjectCollection
    oControlPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 0 To 3
        Call oControlPoints.Add(oPosition(i))
    Next

    ' Create teh revision cloud definition object.
    Dim oRCDef As RevisionCloudDefinition
    oRCDef = oActiveSheet.RevisionClouds.CreateRevisionCloudDefinition(oControlPoints)

    ' Create the revision cloud.
    Dim oRevisionCloud As RevisionCloud
    oRevisionCloud = oActiveSheet.RevisionClouds.Add(oRCDef)
```

Query revision table

Description

This sample illustrates querying the contents of the revision table.

Code Samples

- VBA
- iLogic

To run this sample have a sheet active that contains a revision table.

Copy Code

```
Public Sub RevisionTableQuery()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first revision table on the active sheet.
    ' This assumes that a revision table is on the active sheet.
    Dim oRevTable As RevisionTable
    Set oRevTable = oDrawDoc.ActiveSheet.RevisionTables.Item(1)

    ' Iterate through the contents of the revision table.
    Dim i As Long
    For i = 1 To oRevTable.RevisionTableRows.Count
        ' Get the current row.
        Dim oRow As RevisionTableRow
        Set oRow = oRevTable.RevisionTableRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oRevTable.RevisionTableColumns.Count
            ' Get the current cell.
            Dim oCell As RevisionTableCell
            Set oCell = oRow.Item(j)

            ' Display the value of the current cell.
            Debug.Print "Row: " & i & ", Column: " & oRevTable.RevisionTableColumns.Item(j).Title & " = "; oCell.Text
        Next
    Next
End Sub
```

To run this sample have a sheet active that contains a revision table.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first revision table on the active sheet.
    ' This assumes that a revision table is on the active sheet.
    Dim oRevTable As RevisionTable
    oRevTable = oDrawDoc.ActiveSheet.RevisionTables.Item(1)

    ' Iterate through the contents of the revision table.
    Dim i As Long
    For i = 1 To oRevTable.RevisionTableRows.Count
        ' Get the current row.
        Dim oRow As RevisionTableRow
        oRow = oRevTable.RevisionTableRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oRevTable.RevisionTableColumns.Count
            ' Get the current cell.
            Dim oCell As RevisionTableCell
            oCell = oRow.Item(j)

            ' Display the value of the current cell.
            Logger.Info("Row: " & i & ", Column: " & oRevTable.RevisionTableColumns.Item(j).Title & " = " & oCell.Text)

        Next
    Next
```

Create SketchedSymbol Definition

Description

This sample illustrates creating a new sketched symbol definition object and inserting it into the active sheet.

Code Samples

- VBA
- iLogic

This sample consists of two subs. The first demonstrates the creation of a sketched symbol definition and the second inserts it into the active sheet. To run the sample have a drawing document open and run the CreateSketchedSymbolDefinition Sub. After this you can run the InsertSketchedSymbolOnSheet to insert the sketched symbol into the active sheet. The insertion sub demonstrates the use of the insertion point in the symbol's definition while inserting the symbol.

Copy Code

```
Public Sub CreateSketchedSymbolDefinition()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new sketched symbol definition.
    Dim oSketchedSymbolDef As SketchedSymbolDefinition
    Set oSketchedSymbolDef = oDrawDoc.SketchedSymbolDefinitions.Add("Circular Callout")
```

```
    ' Open the sketched symbol definition's sketch for edit. This is done by calling the Edit
    ' method of the SketchedSymbolDefinition to obtain a DrawingSketch. This actually creates
    ' a copy of the sketched symbol definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oSketchedSymbolDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add sketched symbol graphics.
    Dim oSketchLine As SketchLine
    Set oSketchLine = oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(20, 0))

    Dim oSketchCircle As SketchCircle
    Set oSketchCircle = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(22, 0), 2)

    Call oSketch.GeometricConstraints.AddCoincident(oSketchLine.EndSketchPoint, oSketchCircle)

    ' Make the starting point of the sketch line the insertion point
    oSketchLine.StartSketchPoint.InsertionPoint = True

    ' Add a prompted text field at the center of the sketch circle.
    Dim sText As String
    sText = "<Prompt>Enter text 1</Prompt>"
    Dim oTextBox As TextBox
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(22, 0), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    Call oSketchedSymbolDef.ExitEdit(True)
End Sub

Public Sub InsertSketchedSymbolOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired sketched symbol definition.
    Dim oSketchedSymbolDef As SketchedSymbolDefinition
    Set oSketchedSymbolDef = oDrawDoc.SketchedSymbolDefinitions.Item("Circular Callout")

    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' This sketched symbol definition contains one prompted string input. An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(0) As String
    sPromptStrings(0) = "A"

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Add an instance of the sketched symbol definition to the sheet.
    ' Rotate the instance by 45 degrees and scale by .75 when adding.
    ' The symbol will be inserted at (0,0) on the sheet. Since the
    ' start point of the line was marked as the insertion point, the
    ' start point should end up at (0,0).
    Dim oSketchedSymbol As SketchedSymbol
    Set oSketchedSymbol = oSheet.SketchedSymbols.Add(oSketchedSymbolDef, oTG.CreatePoint2d(0, 0), (3.14159 / 4), 0.75, sPromptStrings)
End Sub
```

This sample consists of two subs. The first demonstrates the creation of a sketched symbol definition and the second inserts it into the active sheet. To run the sample have a drawing document open and run the CreateSketchedSymbolDefinition Sub. After this you can run the InsertSketchedSymbolOnSheet to insert the sketched symbol into the active sheet. The insertion sub demonstrates the use of the insertion point in the symbol's definition while inserting the symbol.

Copy Code

```
Sub Main
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new sketched symbol definition.
    Dim oSketchedSymbolDef As SketchedSymbolDefinition
    oSketchedSymbolDef = oDrawDoc.SketchedSymbolDefinitions.Add("Circular Callout")

    ' Open the sketched symbol definition's sketch for edit. This is done by calling the Edit
    ' method of the SketchedSymbolDefinition to obtain a DrawingSketch. This actually creates
    ' a copy of the sketched symbol definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oSketchedSymbolDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add sketched symbol graphics.
    Dim oSketchLine As SketchLine
    oSketchLine = oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(20, 0))

    Dim oSketchCircle As SketchCircle
    oSketchCircle = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(22, 0), 2)

    Call oSketch.GeometricConstraints.AddCoincident(oSketchLine.EndSketchPoint, oSketchCircle)

    ' Make the starting point of the sketch line the insertion point
    oSketchLine.StartSketchPoint.InsertionPoint = True

    ' Add a prompted text field at the center of the sketch circle.
    Dim sText As String
    sText = "<Prompt>Enter text 1</Prompt>"
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(22, 0), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter
```

```
    Call oSketchedSymbolDef.ExitEdit(True)
End Sub

Public Sub InsertSketchedSymbolOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired sketched symbol definition.
    Dim oSketchedSymbolDef As SketchedSymbolDefinition
    oSketchedSymbolDef = oDrawDoc.SketchedSymbolDefinitions.Item("Circular Callout")

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' This sketched symbol definition contains one prompted string input. An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(0) As String
    sPromptStrings(0) = "A"

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Add an instance of the sketched symbol definition to the sheet.
    ' Rotate the instance by 45 degrees and scale by .75 when adding.
    ' The symbol will be inserted at (0,0) on the sheet. Since the
    ' start point of the line was marked as the insertion point, the
    ' start point should end up at (0,0).
    Dim oSketchedSymbol As SketchedSymbol
    oSketchedSymbol = oSheet.SketchedSymbols.Add(oSketchedSymbolDef, oTG.CreatePoint2d(0, 0), (3.14159 / 4), 0.75, sPromptStrings)
End Sub
```

Create sketched symbol and leader

Description

This sample illustrates creating sketched symbol with a leader.

Code Samples

- VBA
- iLogic

Before running this sample, open a drawing document, create a sketched symbol definition and select a linear drawing edge. The sample uses the first sketched symbol definition in the document to create the symbol.

Copy Code

```
Public Sub AddSymbolWithLeader()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Set oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    Set oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    Set oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)

    ' Get the first symbol definition
    Dim oSketchSymDef As SketchedSymbolDefinition
    Set oSketchSymDef = oDrawDoc.SketchedSymbolDefinitions.Item(1)

    ' Create the symbol with a leader
    Dim oSketchedSymbol As SketchedSymbol
```

```
    Set oSketchedSymbol = oActiveSheet.SketchedSymbols.AddWithLeader(oSketchSymDef, oLeaderPoints)
End Sub
```

Before running this sample, open a drawing document, create a sketched symbol definition and select a linear drawing edge. The sample uses the first sketched symbol definition in the document to create the symbol.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear
    Dim oMidPoint As Point2d
    oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve)

    Call oLeaderPoints.Add(oGeometryIntent)

    ' Get the first symbol definition
    Dim oSketchSymDef As SketchedSymbolDefinition
    oSketchSymDef = oDrawDoc.SketchedSymbolDefinitions.Item(1)

    ' Create the symbol with a leader
    Dim oSketchedSymbol As SketchedSymbol
    oSketchedSymbol = oActiveSheet.SketchedSymbols.AddWithLeader(oSketchSymDef, oLeaderPoints)
```

Add surface texture symbol to dimension

Description

This sample demonstrates the creation of a surface texture symbol attached to the extension line of a drawing dimension.

Code Samples

- VBA
- iLogic

Select a linear general dimension in a drawing and run the sample.

Copy Code

```
Public Sub AddSurfaceTextureSymbol()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Check to make sure a linear dimension is selected.
    If Not TypeOf oDrawDoc.SelectSet.Item(1) Is LinearGeneralDimension Then
        MsgBox "A linear general dimension must be selected."
        Exit Sub
    End If

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing dimension.
    ' This assumes that a linear general dimension is selected.
    Dim oLinearDim As LinearGeneralDimension
    Set oLinearDim = oDrawDoc.SelectSet.Item(1)

    ' Get the mid point of the first extension line of the dimension
    Dim oMidPoint As Object
    Set oMidPoint = oLinearDim.ExtensionLineOne.MidPoint

    ' Set a reference to the TransientGeometry object.
```

```
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oLinearDim, oMidPoint)

    Call oLeaderPoints.Add(oGeometryIntent)

    ' Create the symbol with a leader
    Dim oSymbol As SurfaceTextureSymbol
    Set oSymbol = oActiveSheet.SurfaceTextureSymbols.Add(oLeaderPoints, _
            kMaterialRemovalRequiredSurfaceType, _
            True, _
            True, _
            True, _
            0.1, _
            , , , , , _
            kParticulateNondirectional)
End Sub
```

Select a linear general dimension in a drawing and run the sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Check to make sure a linear dimension is selected.
    If Not TypeOf oDrawDoc.SelectSet.Item(1) Is LinearGeneralDimension Then
        MsgBox("A linear general dimension must be selected.")
        Exit Sub
    End If

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing dimension.
    ' This assumes that a linear general dimension is selected.
    Dim oLinearDim As LinearGeneralDimension
    oLinearDim = oDrawDoc.SelectSet.Item(1)

    ' Get the mid point of the first extension line of the dimension
    Dim oMidPoint As Object
    oMidPoint = oLinearDim.ExtensionLineOne.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a few leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Create an intent and add to the leader points collection.
    ' This is the geometry that the symbol will attach to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oLinearDim, oMidPoint)

    Call oLeaderPoints.Add(oGeometryIntent)

    ' Create the symbol with a leader
    Dim oSymbol As SurfaceTextureSymbol
    oSymbol = oActiveSheet.SurfaceTextureSymbols.Add(oLeaderPoints, _
            kMaterialRemovalRequiredSurfaceType, _
            True, _
            True, _
            True, _
            0.1, _
            , , , , , _
            kParticulateNondirectional)
```

Title Block Definition Create and Insert

Description

This sample illustrates creating a new title block definition object and inserting it into the active sheet. This sample consists of two subs. The first demonstrates the creation of a title block definition and the second inserts it into the active sheet.

Code Samples

- VBA
- iLogic

To run the sample have a drawing document open and run the CreateTitleBlockDefinition Sub. After this you can run the InsertTitleBlockOnSheet to insert the title block into the active sheet.

Copy Code

```
Public Sub CreateTitleBlockDefinition()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new title block defintion.
    Dim oTitleBlockDef As TitleBlockDefinition
    Set oTitleBlockDef = oDrawDoc.TitleBlockDefinitions.Add("Sample Title Block")

    ' Open the title block definition's sketch for edit.  This is done by calling the Edit
    ' method of the TitleBlockDefinition to obtain a DrawingSketch.  This actually creates
    ' a copy of the title block definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oTitleBlockDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add title block graphics.
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(9, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 1.5), oTG.CreatePoint2d(9, 1.5))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 2.25), oTG.CreatePoint2d(9, 2.25))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(4.5, 1.5), oTG.CreatePoint2d(4.5, 2.25))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(3, 2.25), oTG.CreatePoint2d(3, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(6, 2.25), oTG.CreatePoint2d(6, 3))

    ' Add some static text to the title block.
    Dim sText As String
    sText = "TITLE BLOCK"
    Dim oTextBox As TextBox
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4.5, 0.75), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    sText = "Static Text"
    Set oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(0, 1.5), oTG.CreatePoint2d(4.5, 2.25), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add some prompted text fields.
    sText = "<Prompt>Enter text 1</Prompt>"
    Set oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(4.5, 1.5), oTG.CreatePoint2d(9, 2.25), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

     sText = "<Prompt>Enter text 2</Prompt>"
    Set oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(0, 2.25), oTG.CreatePoint2d(3, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add some property text.
    ' Add the property value of Author from the drawing
    sText = "<Property Document='drawing' FormatID='{F29F85E0-4FF9-1068-AB91-08002B27B3D9}' PropertyID='4' />"
    Set oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(3, 2.25), oTG.CreatePoint2d(6, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add the property value of Subject from the drawing
    sText = "<Property Document='drawing' FormatID='{F29F85E0-4FF9-1068-AB91-08002B27B3D9}' PropertyID='3' />"
    Set oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(6, 2.25), oTG.CreatePoint2d(9, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    Call oTitleBlockDef.ExitEdit(True)
End Sub

Public Sub InsertTitleBlockOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired border defintion.
    Dim oTitleBlockDef As TitleBlockDefinition
    Set oTitleBlockDef = oDrawDoc.TitleBlockDefinitions.Item("Sample Title Block")

    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a title block and delete it if it does.
    If Not oSheet.TitleBlock Is Nothing Then
        oSheet.TitleBlock.Delete
    End If

    ' This title block definition contains one prompted string input.  An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(1 To 2) As String
    sPromptStrings(1) = "String 1"
    sPromptStrings(2) = "String 2"

    ' Add an instance of the title block definition to the sheet.
    Dim oTitleBlock As TitleBlock
    Set oTitleBlock = oSheet.AddTitleBlock(oTitleBlockDef, , sPromptStrings)
End Sub
```

To run the sample have a drawing document open and run the CreateTitleBlockDefinition Sub. After this you can run the InsertTitleBlockOnSheet to insert the title block into the active sheet.

Copy Code

```
Sub Main
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new title block defintion.
    Dim oTitleBlockDef As TitleBlockDefinition
    oTitleBlockDef = oDrawDoc.TitleBlockDefinitions.Add("Sample Title Block")

    ' Open the title block definition's sketch for edit.  This is done by calling the Edit
    ' method of the TitleBlockDefinition to obtain a DrawingSketch.  This actually creates
    ' a copy of the title block definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oTitleBlockDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add title block graphics.
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(9, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 1.5), oTG.CreatePoint2d(9, 1.5))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 2.25), oTG.CreatePoint2d(9, 2.25))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(4.5, 1.5), oTG.CreatePoint2d(4.5, 2.25))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(3, 2.25), oTG.CreatePoint2d(3, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(6, 2.25), oTG.CreatePoint2d(6, 3))

    ' Add some static text to the title block.
    Dim sText As String
    sText = "TITLE BLOCK"
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4.5, 0.75), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    sText = "Static Text"
    oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(0, 1.5), oTG.CreatePoint2d(4.5, 2.25), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add some prompted text fields.
    sText = "<Prompt>Enter text 1</Prompt>"
    oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(4.5, 1.5), oTG.CreatePoint2d(9, 2.25), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

     sText = "<Prompt>Enter text 2</Prompt>"
    oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(0, 2.25), oTG.CreatePoint2d(3, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add some property text.
    ' Add the property value of Author from the drawing
    sText = "<Property Document='drawing' FormatID='{F29F85E0-4FF9-1068-AB91-08002B27B3D9}' PropertyID='4' />"
    oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(3, 2.25), oTG.CreatePoint2d(6, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    ' Add the property value of Subject from the drawing
    sText = "<Property Document='drawing' FormatID='{F29F85E0-4FF9-1068-AB91-08002B27B3D9}' PropertyID='3' />"
    oTextBox = oSketch.TextBoxes.AddByRectangle(oTG.CreatePoint2d(6, 2.25), oTG.CreatePoint2d(9, 3), sText)
    oTextBox.VerticalJustification = kAlignTextMiddle
    oTextBox.HorizontalJustification = kAlignTextCenter

    Call oTitleBlockDef.ExitEdit(True)
End Sub

Public Sub InsertTitleBlockOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired border defintion.
    Dim oTitleBlockDef As TitleBlockDefinition
    oTitleBlockDef = oDrawDoc.TitleBlockDefinitions.Item("Sample Title Block")

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a title block and delete it if it does.
    If Not oSheet.TitleBlock Is Nothing Then
        oSheet.TitleBlock.Delete
    End If

    ' This title block definition contains one prompted string input.  An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(0 To 1) As String
    sPromptStrings(0) = "String 1"
    sPromptStrings(1) = "String 2"

    ' Add an instance of the title block definition to the sheet.
    Dim oTitleBlock As TitleBlock
    oTitleBlock = oSheet.AddTitleBlock(oTitleBlockDef, , sPromptStrings)
End Sub
```

Copying a title block definition

Description

This sample demonstrates copying a title block definition from one drawing to another and replacing the existing title blocks in the drawing with the new title block.

Code Samples

- VBA
- iLogic

Before running this sample, have the drawing document open with a sheet active that contains the title block you want to copy. Also, change the path of the new document (oNewDocument) to point to a drawing document in which you want the titleblocks changed.

Copy Code

```
Public Sub TitleBlockCopy()
    Dim oSourceDocument As DrawingDocument
    Set oSourceDocument = ThisApplication.ActiveDocument

    ' Open the new drawing to copy the title block into.
    Dim oNewDocument As DrawingDocument
    Set oNewDocument = ThisApplication.Documents.Open("C:\temp\TitleBlockChange.idw")

    ' Get the new source title block definition.
    Dim oSourceTitleBlockDef As TitleBlockDefinition
    Set oSourceTitleBlockDef = oSourceDocument.ActiveSheet.TitleBlock.Definition

    ' Get the new title block definition.
    Dim oNewTitleBlockDef As TitleBlockDefinition
    Set oNewTitleBlockDef = oSourceTitleBlockDef.CopyTo(oNewDocument)

    ' Iterate through the sheets.
    Dim oSheet As Sheet
    For Each oSheet In oNewDocument.Sheets
        oSheet.Activate

        oSheet.TitleBlock.Delete
        Call oSheet.AddTitleBlock(oNewTitleBlockDef)
    Next
End Sub
```

Before running this sample, have the drawing document open with a sheet active that contains the title block you want to copy. Also, change the path of the new document (oNewDocument) to point to a drawing document in which you want the titleblocks changed.

Copy Code

```
    Dim oSourceDocument As DrawingDocument
    oSourceDocument = ThisApplication.ActiveDocument

    ' Open the new drawing to copy the title block into.
    Dim oNewDocument As DrawingDocument
    oNewDocument = ThisApplication.Documents.Open("C:\temp\TitleBlockChange.idw")

    ' Get the new source title block definition.
    Dim oSourceTitleBlockDef As TitleBlockDefinition
    oSourceTitleBlockDef = oSourceDocument.ActiveSheet.TitleBlock.Definition

    ' Get the new title block definition.
    Dim oNewTitleBlockDef As TitleBlockDefinition
    oNewTitleBlockDef = oSourceTitleBlockDef.CopyTo(oNewDocument)

    ' Iterate through the sheets.
    Dim oSheet As Sheet
    For Each oSheet In oNewDocument.Sheets
        oSheet.Activate

        oSheet.TitleBlock.Delete
        Call oSheet.AddTitleBlock(oNewTitleBlockDef)
    Next
```

AutoCAD block definitions import

Description

This sample demonstrates importing AutoCAD block definitions from an external dwg file.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub ImportAutoCADBlockDefinitionsFromFile()
    ' Set a reference to the drawing document.
    ' This assumes an Inventor dwg drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Create an array of names of the definitions to import
    Dim oDefNames(1) As String
    oDefNames(0) = "DatumFilled45"
    oDefNames(1) = "Borders Default Border"

    ' Import definitions from an external dwg file
    ' Only the specified definitions will be imported. If you want all definitions
    ' to be imported, you can leave the 2nd argument as unspecified.
    Dim oBlockDefs As AutoCADBlockDefinitionsEnumerator
    Set oBlockDefs = oDrawDoc.AutoCADBlockDefinitions.AddFromFile("C:\temp\acad.dwg", oDefNames)
End Sub
```

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes an Inventor dwg drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create an array of names of the definitions to import
    Dim oDefNames(1) As String
    oDefNames(0) = "DatumFilled45"
    oDefNames(1) = "Borders Default Border"

    ' Import definitions from an external dwg file
    ' Only the specified definitions will be imported. If you want all definitions
    ' to be imported, you can leave the 2nd argument as unspecified.
    Dim oBlockDefs As AutoCADBlockDefinitionsEnumerator
    oBlockDefs = oDrawDoc.AutoCADBlockDefinitions.AddFromFile("C:\temp\acad.dwg", oDefNames)
```

AutoCAD block insertion

Description

Demonstrates inserting an AutoCAD block.

Code Samples

- VBA
- iLogic

Open an Inventor dwg file and run the following sample.

Copy Code

```
Public Sub InsertAutoCADBlockOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired AutoCAD block definition.
    Dim oBlockDef As AutoCADBlockDefinition
    Set oBlockDef = oDrawDoc.AutoCADBlockDefinitions.Item("Filled-1")

    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' If the definition contains prompted string inputs...
    'Dim sPromptStrings(1) As String
    'sPromptStrings(0) = "String 1"
    'sPromptStrings(1) = "String 2"

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Add an instance of the block definition to the sheet.
    ' Rotate the instance by 45 degrees and scale by .75 when adding.
    ' The symbol will be inserted at (10,10) on the sheet.
    Dim oAutoCADBlock As AutoCADBlock
    Set oAutoCADBlock = oSheet.AutoCADBlocks.Add(oBlockDef, oTG.CreatePoint2d(10, 10), (3.14159 / 4), 0.75)
End Sub
```

Open an Inventor dwg file and run the following sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired AutoCAD block definition.
    Dim oBlockDef As AutoCADBlockDefinition
    oBlockDef = oDrawDoc.AutoCADBlockDefinitions.Item("Filled-1")

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' If the definition contains prompted string inputs...
    'Dim sPromptStrings(1) As String
    'sPromptStrings(0) = "String 1"
    'sPromptStrings(1) = "String 2"

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Add an instance of the block definition to the sheet.
    ' Rotate the instance by 45 degrees and scale by .75 when adding.
    ' The symbol will be inserted at (10,10) on the sheet.
    Dim oAutoCADBlock As AutoCADBlock
    oAutoCADBlock = oSheet.AutoCADBlocks.Add(oBlockDef, oTG.CreatePoint2d(10, 10), (3.14159 / 4), 0.75)
```

Balloons - edit

Description

This sample demonstrates the editing of balloons in a drawing.

Code Samples

- VBA
- iLogic

To run this sample, open a drawing document with the active sheet that has several balloons (including attached balloons) placed on it. This sample modifies the type and values of all the balloons.

Copy Code

```
Public Sub EditBalloons()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    Dim BalloonCount As Long
    BalloonCount = 1

    Dim oBalloon As Balloon

    ' Iterate over each balloon on the sheet.
    For Each oBalloon In oSheet.Balloons

        ' Change the balloon type
        oBalloon.SetBalloonType (kHexagonBalloonType)

        ' Change balloon placement direction so all the attached
        ' balloons are placed to the right of the previous ones.
        oBalloon.PlacementDirection = kBottomDirection

        Dim ValueSetCount As Long
        ValueSetCount = 1

        Dim oBalloonValueSet As BalloonValueSet

        ' Iterate over each value set (attached balloons) in a balloon.
        For Each oBalloonValueSet In oBalloon.BalloonValueSets

            ' Change the override value.
            oBalloonValueSet.OverrideValue = "Balloon" & BalloonCount & ": ValueSet" & ValueSetCount
            ValueSetCount = ValueSetCount + 1
        Next
        BalloonCount = BalloonCount + 1
    Next
End Sub
```

To run this sample, open a drawing document with the active sheet that has several balloons (including attached balloons) placed on it. This sample modifies the type and values of all the balloons.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    Dim BalloonCount As Long
    BalloonCount = 1

    Dim oBalloon As Balloon

    ' Iterate over each balloon on the sheet.
    For Each oBalloon In oSheet.Balloons

        ' Change the balloon type
        oBalloon.SetBalloonType (kHexagonBalloonType)

        ' Change balloon placement direction so all the attached
        ' balloons are placed to the right of the previous ones.
        oBalloon.PlacementDirection = kBottomDirection

        Dim ValueSetCount As Long
        ValueSetCount = 1

        Dim oBalloonValueSet As BalloonValueSet

        ' Iterate over each value set (attached balloons) in a balloon.
        For Each oBalloonValueSet In oBalloon.BalloonValueSets

            ' Change the override value.
            oBalloonValueSet.OverrideValue = "Balloon" & BalloonCount & ": ValueSet" & ValueSetCount
            ValueSetCount = ValueSetCount + 1
        Next
        BalloonCount = BalloonCount + 1
    Next
```

Creation a balloon

Description

This sample demonstrates the creation of a balloon.

Code Samples

- VBA
- iLogic

Select a linear drawing curve and run the sample.

Copy Code

```
Public Sub CreateBalloon()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    Set oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    Set oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear. This is used to calculate the position of balloon and leader line.
    Dim oMidPoint As Point2d
    Set oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    Set oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a couple of leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Add the GeometryIntent to the leader points collection.
    ' This is the geometry that the balloon will attach to. The mid-point of the geometry will be attached to.
    Dim oGeometryIntent As GeometryIntent
    Set oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve,oMidPoint)
    Call oLeaderPoints.Add(oGeometryIntent)

    ' Set a reference to the parent drawing view of the selected curve
    Dim oDrawingView As DrawingView
    Set oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the referenced model document
    Dim oModelDoc As Document
    Set oModelDoc = oDrawingView.ReferencedDocumentDescriptor.ReferencedDocument

    ' Check if a partslist or a balloon has already been created for this model
    Dim IsDrawingBOMDefined As Boolean
    IsDrawingBOMDefined = oDrawDoc.DrawingBOMs.IsDrawingBOMDefined(oModelDoc.FullFileName)

    Dim oBalloon As Balloon

    If IsDrawingBOMDefined Then

        ' Just create the balloon with the leader points
        ' All other arguments can be ignored
        Set oBalloon = oDrawDoc.ActiveSheet.Balloons.Add(oLeaderPoints)
    Else

        ' First check if the 'structured' BOM view has been enabled in the model

        ' Set a reference to the model's BOM object
        Dim oBOM As BOM
        Set oBOM = oModelDoc.ComponentDefinition.BOM

        If oBOM.StructuredViewEnabled Then

            ' Level needs to be specified
            ' Numbering options have already been defined
            ' Get the Level ('All levels' or 'First level only')
            ' from the model BOM view - must use the same here
            Dim Level As PartsListLevelEnum
            If oBOM.StructuredViewFirstLevelOnly Then
                Level = kStructured
            Else
                Level = kStructuredAllLevels
            End If

            ' Create the balloon by specifying just the level
            Set oBalloon = oActiveSheet.Balloons.Add(oLeaderPoints, , Level)
```

```
        Else

            ' Level and numbering options must be specified
            ' The corresponding model BOM view will automatically be enabled
            Dim oNumberingScheme As NameValueMap
            Set oNumberingScheme = ThisApplication.TransientObjects.CreateNameValueMap

            ' Add the option for a comma delimiter
            oNumberingScheme.Add "Delimiter", ","

            ' Create the balloon by specifying the level and numbering scheme
            Set oBalloon = oActiveSheet.Balloons.Add(oLeaderPoints, , kStructuredAllLevels, oNumberingScheme)
        End If
    End If
End Sub
```

Select a linear drawing curve and run the sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing curve segment.
    ' This assumes that a drwaing curve is selected.
    Dim oDrawingCurveSegment As DrawingCurveSegment
    oDrawingCurveSegment = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the drawing curve.
    Dim oDrawingCurve As DrawingCurve
    oDrawingCurve = oDrawingCurveSegment.Parent

    ' Get the mid point of the selected curve
    ' assuming that the selection curve is linear. This is used to calculate the position of balloon and leader line.
    Dim oMidPoint As Point2d
    oMidPoint = oDrawingCurve.MidPoint

    ' Set a reference to the TransientGeometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oLeaderPoints As ObjectCollection
    oLeaderPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Create a couple of leader points.
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 10))
    Call oLeaderPoints.Add(oTG.CreatePoint2d(oMidPoint.X + 10, oMidPoint.Y + 5))

    ' Add the GeometryIntent to the leader points collection.
    ' This is the geometry that the balloon will attach to. The mid-point of the geometry will be attached to.
    Dim oGeometryIntent As GeometryIntent
    oGeometryIntent = oActiveSheet.CreateGeometryIntent(oDrawingCurve,oMidPoint)
    Call oLeaderPoints.Add(oGeometryIntent)

    ' Set a reference to the parent drawing view of the selected curve
    Dim oDrawingView As DrawingView
    oDrawingView = oDrawingCurve.Parent

    ' Set a reference to the referenced model document
    Dim oModelDoc As Document
    oModelDoc = oDrawingView.ReferencedDocumentDescriptor.ReferencedDocument

    ' Check if a partslist or a balloon has already been created for this model
    Dim IsDrawingBOMDefined As Boolean
    IsDrawingBOMDefined = oDrawDoc.DrawingBOMs.IsDrawingBOMDefined(oModelDoc.FullFileName)

    Dim oBalloon As Balloon

    If IsDrawingBOMDefined Then

        ' Just create the balloon with the leader points
        ' All other arguments can be ignored
        oBalloon = oDrawDoc.ActiveSheet.Balloons.Add(oLeaderPoints)
    Else

        ' First check if the 'structured' BOM view has been enabled in the model

        ' Set a reference to the model's BOM object
        Dim oBOM As BOM
        oBOM = oModelDoc.ComponentDefinition.BOM

        If oBOM.StructuredViewEnabled Then

            ' Level needs to be specified
            ' Numbering options have already been defined
            ' Get the Level ('All levels' or 'First level only')
            ' from the model BOM view - must use the same here
            Dim Level As PartsListLevelEnum
            If oBOM.StructuredViewFirstLevelOnly Then
                Level = kStructured
            Else
                Level = kStructuredAllLevels
            End If

            ' Create the balloon by specifying just the level
            oBalloon = oActiveSheet.Balloons.Add(oLeaderPoints, , Level)
        Else

            ' Level and numbering options must be specified
            ' The corresponding model BOM view will automatically be enabled
            Dim oNumberingScheme As NameValueMap
            oNumberingScheme = ThisApplication.TransientObjects.CreateNameValueMap
```

```
            ' Add the option for a comma delimiter
            oNumberingScheme.Add("Delimiter", ",")

            ' Create the balloon by specifying the level and numbering scheme
            oBalloon = oActiveSheet.Balloons.Add(oLeaderPoints, , kStructuredAllLevels, oNumberingScheme)
        End If
    End If
```

Find component referenced by balloon

Description

This sample demonstrates how to find the component that a balloon references.

Code Samples

- VBA
- iLogic

Select a balloon in a drawing and run the following sample.

Copy Code

```
Public Sub GetComponentReferencedByBalloon()
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Get the selected balloon
    Dim oBalloon As Balloon
    Set oBalloon = oDoc.SelectSet(1)

    Dim oBalloonValueSet As BalloonValueSet
    For Each oBalloonValueSet In oBalloon.BalloonValueSets
        Dim strDisplay As String
        strDisplay = "Balloon Item Number: "

        ' Add the balloon item number (the overridden value if there is one)
        If oBalloonValueSet.OverrideValue = "" Then
            strDisplay = strDisplay & oBalloonValueSet.Value
        Else
            strDisplay = strDisplay & oBalloonValueSet.OverrideValue
        End If

        Dim oDrawingBOMRow As DrawingBOMRow
        Set oDrawingBOMRow = oBalloonValueSet.ReferencedRow

        If oDrawingBOMRow.Custom Then
            ' The referenced item is a custom parts list row.
            strDisplay = strDisplay & vbNewLine & "Referenced Component(s):"
            strDisplay = strDisplay & vbNewLine & "     Custom PartsList Row"
        Else
            Dim oBOMRow As BOMRow
            Set oBOMRow = oDrawingBOMRow.BOMRow

            ' Add the Item Number from the model BOM.
            strDisplay = strDisplay & vbNewLine & "BOM Item Number: " & oBOMRow.ItemNumber

            strDisplay = strDisplay & vbNewLine & "Referenced Component(s):"

            Dim oCompDefs As ComponentDefinitionsEnumerator
            Set oCompDefs = oBOMRow.ComponentDefinitions

            If oDrawingBOMRow.Virtual Then
                ' The referenced item is a virtual component.
                strDisplay = strDisplay & vbNewLine & "     Virtual: " & oCompDefs.Item(1).DisplayName
            Else

                ' Add the document name of the referenced component.
                ' There could be multiple if the balloon references
                ' a merged BOM row in the model.
                Dim oCompDef As ComponentDefinition
                For Each oCompDef In oCompDefs
                    strDisplay = strDisplay & vbNewLine & "     " & oCompDef.Document.FullDocumentName
                Next

            End If
        End If
        MsgBox strDisplay
    Next
End Sub
```

Select a balloon in a drawing and run the following sample.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Get the selected balloon
    Dim oBalloon As Balloon
    oBalloon = oDoc.SelectSet(1)

    Dim oBalloonValueSet As BalloonValueSet
    For Each oBalloonValueSet In oBalloon.BalloonValueSets
        Dim strDisplay As String
        strDisplay = "Balloon Item Number: "
```

```
        ' Add the balloon item number (the overridden value if there is one)
        If oBalloonValueSet.OverrideValue = "" Then
            strDisplay = strDisplay & oBalloonValueSet.Value
        Else
            strDisplay = strDisplay & oBalloonValueSet.OverrideValue
        End If

        Dim oDrawingBOMRow As DrawingBOMRow
        oDrawingBOMRow = oBalloonValueSet.ReferencedRow

        If oDrawingBOMRow.Custom Then
            ' The referenced item is a custom parts list row.
            strDisplay = strDisplay & vbNewLine & "Referenced Component(s):"
            strDisplay = strDisplay & vbNewLine & "     Custom PartsList Row"
        Else
            Dim oBOMRow As BOMRow
            oBOMRow = oDrawingBOMRow.BOMRow

            ' Add the Item Number from the model BOM.
            strDisplay = strDisplay & vbNewLine & "BOM Item Number: " & oBOMRow.ItemNumber

            strDisplay = strDisplay & vbNewLine & "Referenced Component(s):"

            Dim oCompDefs As ComponentDefinitionsEnumerator
            oCompDefs = oBOMRow.ComponentDefinitions

            If oDrawingBOMRow.Virtual Then
                ' The referenced item is a virtual component.
                strDisplay = strDisplay & vbNewLine & "     Virtual: " & oCompDefs.Item(1).DisplayName
            Else

                ' Add the document name of the referenced component.
                ' There could be multiple if the balloon references
                ' a merged BOM row in the model.
                Dim oCompDef As ComponentDefinition
                For Each oCompDef In oCompDefs
                    strDisplay = strDisplay & vbNewLine & "     " & oCompDef.Document.FullDocumentName
                Next

            End If
        End If
        MsgBox(strDisplay)
    Next
```

Create linear foreshortened dimension sample

Description

This sample demonstrates the creation of a linear foreshortened dimension.

Code Samples

- VBA
- iLogic

Have a drawing document open and select two paralell linear drawing curve segments in the same drawing view and run the following macro.

Copy Code

```
Sub CreateLinearForeshortenedDimSample()
    ' Open a drawing document and select two parallel linear curves in same drawing view first.
    Dim oApp As Inventor.Application
    Set oApp = ThisApplication

    Dim oDoc As Document
    Set oDoc = oApp.ActiveDocument
    If Not (oDoc Is Nothing) Then
        If Not (oDoc.DocumentType = kDrawingDocumentObject) Then
            MsgBox "Please activate a drawing document for this sample.", vbCritical, "Inventor"
            Exit Sub
        End If
    Else
        MsgBox "Please open a drawing document for this sample.", vbCritical, "Inventor"
        Exit Sub
    End If

    If Not (oDoc.SelectSet.Count = 2) Then
        MsgBox "Please select two parallel linear drawing curves from same drawing view for this sample.", vbCritical, "Inventor"
        Exit Sub
    ElseIf (oDoc.SelectSet(1).Type <> kDrawingCurveSegmentObject Or oDoc.SelectSet(1).Type <> kDrawingCurveSegmentObject) Then
        MsgBox "Please select two parallel linear drawing curves from same drawing view for this sample.", vbCritical, "Inventor"
        Exit Sub
    End If

    Dim oCurve1 As DrawingCurve, oCurve2 As DrawingCurve
    Set oCurve1 = oDoc.SelectSet(1).Parent
    Set oCurve2 = oDoc.SelectSet(2).Parent

    Dim oSheet As Sheet
    Set oSheet = oCurve1.Parent.Parent

    ' Create two GeometryIntent based on the selected drawing curve segments.
    Dim oIntent1 As GeometryIntent, oIntent2 As GeometryIntent
    Set oIntent1 = oSheet.CreateGeometryIntent(oCurve1)
    Set oIntent2 = oSheet.CreateGeometryIntent(oCurve2)

    Dim oTextPos As Point2d
    Set oTextPos = oApp.TransientGeometry.CreatePoint2d(12, 12)
```

```
    ' Create the linear foreshortened dimension
    Dim oLinearForeshortenedDim As LinearGeneralDimension
    Set oLinearForeshortenedDim = oSheet.DrawingDimensions.GeneralDimensions.AddLinearForeshortened(oTextPos, oIntent1, oIntent2, True

End Sub
```

Have a drawing document open and select two paralell linear drawing curve segments in the same drawing view and run the following macro.

Copy Code

```
    ' Open a drawing document and select two parallel linear curves in same drawing view first.
    Dim oApp As Inventor.Application
    oApp = ThisApplication

    Dim oDoc As Document
    oDoc = oApp.ActiveDocument
    If Not (oDoc Is Nothing) Then
        If Not (oDoc.DocumentType = kDrawingDocumentObject) Then
            MsgBox("Please activate a drawing document for this sample.", vbCritical, "Inventor")
            Exit Sub
        End If
    Else
        MsgBox("Please open a drawing document for this sample.", vbCritical, "Inventor")
        Exit Sub
    End If

    If Not (oDoc.SelectSet.Count = 2) Then
        MsgBox("Please select two parallel linear drawing curves from same drawing view for this sample.", vbCritical, "Inventor")
        Exit Sub
    ElseIf (oDoc.SelectSet(1).Type <> kDrawingCurveSegmentObject Or oDoc.SelectSet(1).Type <> kDrawingCurveSegmentObject) Then
        MsgBox("Please select two parallel linear drawing curves from same drawing view for this sample.", vbCritical, "Inventor")
        Exit Sub
    End If

    Dim oCurve1 As DrawingCurve, oCurve2 As DrawingCurve
    oCurve1 = oDoc.SelectSet(1).Parent
    oCurve2 = oDoc.SelectSet(2).Parent

    Dim oSheet As Sheet
    oSheet = oCurve1.Parent.Parent

    ' Create two GeometryIntent based on the selected drawing curve segments.
    Dim oIntent1 As GeometryIntent, oIntent2 As GeometryIntent
    oIntent1 = oSheet.CreateGeometryIntent(oCurve1)
    oIntent2 = oSheet.CreateGeometryIntent(oCurve2)

    Dim oTextPos As Point2d
    oTextPos = oApp.TransientGeometry.CreatePoint2d(12, 12)

    ' Create the linear foreshortened dimension
    Dim oLinearForeshortenedDim As LinearGeneralDimension
    oLinearForeshortenedDim = oSheet.DrawingDimensions.GeneralDimensions.AddLinearForeshortened(oTextPos, oIntent1, oIntent2, True)
```

Dimensions - edit

Description

This sample demonstrates the editing of sheet dimensions and the associated dimension style.

Code Samples

- VBA
- iLogic

To run this sample, open a drawing document that has one or more drawing views on the active sheet and add various sheet dimensions to it.

Copy Code

```
Public Sub EditDrawingDimensions()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    Dim counter As Long
    counter = 1

    Dim oDrawingDim As DrawingDimension
    For Each oDrawingDim In oSheet.DrawingDimensions

        ' Add some formatted text to all dimensions on the sheet.
        oDrawingDim.Text.FormattedText = " (Metric)"
        counter = counter + 1

    Next

    ' Set a reference to the first general dimension in the collection.
    Dim oGeneralDim As GeneralDimension
    Set oGeneralDim = oSheet.DrawingDimensions.GeneralDimensions.Item(1)

    ' Set a reference to the dimension style of that dimension.
    Dim oDimStyle As DimensionStyle
    Set oDimStyle = oGeneralDim.Style
```

```
    ' Modify some properties of the dimension style.
    ' This will modify all dimensions that use this style.
    oDimStyle.LinearPrecision = kFourDecimalPlacesLinearPrecision
    oDimStyle.AngularPrecision = kFourDecimalPlacesAngularPrecision
    oDimStyle.LeadingZeroDisplay = False
    oDimStyle.Tolerance.SetToSymmetric (0.02)
End Sub
```

To run this sample, open a drawing document that has one or more drawing views on the active sheet and add various sheet dimensions to it.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    Dim counter As Long
    counter = 1

    Dim oDrawingDim As DrawingDimension
    For Each oDrawingDim In oSheet.DrawingDimensions

        ' Add some formatted text to all dimensions on the sheet.
        oDrawingDim.Text.FormattedText = " (Metric)"
        counter = counter + 1

    Next

    ' Set a reference to the first general dimension in the collection.
    Dim oGeneralDim As GeneralDimension
    oGeneralDim = oSheet.DrawingDimensions.GeneralDimensions.Item(1)

    ' Set a reference to the dimension style of that dimension.
    Dim oDimStyle As DimensionStyle
    oDimStyle = oGeneralDim.Style

    ' Modify some properties of the dimension style.
    ' This will modify all dimensions that use this style.
    oDimStyle.LinearPrecision = kFourDecimalPlacesLinearPrecision
    oDimStyle.AngularPrecision = kFourDecimalPlacesAngularPrecision
    oDimStyle.LeadingZeroDisplay = False
    oDimStyle.Tolerance.SetToSymmetric (0.02)
```

Find and remove unattached dimensions

Description

This sample finds and deletes all unattached drawing dimensions on the active sheet in a drawing.

Code Samples

- VBA
- iLogic

Before you run the sample, have a drawing document open that contains several drawing dimensions including unattached (sick) ones.

Copy Code

```
Public Sub DeleteUnattachedDimensions()
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oSheet As Sheet
    Set oSheet = oDoc.ActiveSheet

    Dim oDrawingDim As DrawingDimension

    ' Iterate over all dimensions in the drawing
    ' and delete unattached (sick) dimensions.

    For Each oDrawingDim In oSheet.DrawingDimensions
        If oDrawingDim.Attached = False Then
            Call oDrawingDim.Delete
        End If
    Next
End Sub
```

Before you run the sample, have a drawing document open that contains several drawing dimensions including unattached (sick) ones.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oSheet As Sheet
    oSheet = oDoc.ActiveSheet

    Dim oDrawingDim As DrawingDimension

    ' Iterate over all dimensions in the drawing
```

```
    ' and delete unattached (sick) dimensions.

    For Each oDrawingDim In oSheet.DrawingDimensions
        If oDrawingDim.Attached = False Then
            Call oDrawingDim.Delete
        End If
    Next
```

Aligning drawing dimensions

Description

This sample demonstrates aligning the selected drawing dimensions along a horizontal or vertical axis.

Code Samples

- VBA
- iLogic

The first dimension selected defines the origin of the axis. A drawing document must be open and at least two dimensions selected.

Copy Code

```
Public Sub DimensionAlign()
   Dim oDrawDoc As DrawingDocument
   Set oDrawDoc = ThisApplication.ActiveDocument

   ' Determine if there are any dimensions in the select set.
   Dim oSelectSet As SelectSet
   Set oSelectSet = oDrawDoc.SelectSet
   Dim colDimensions As New Collection
   Dim i As Long
   For i = 1 To oSelectSet.Count
      If Typeof oSelectSet.Item(i) Is DrawingDimension Then
         ' Add any dimensions to the collection. We need to save them
         ' in something besides the selection set because once we start
         ' manipulating them the select set will be cleared.
         colDimensions.Add oSelectSet.Item(i)
      End If
   Next

   If colDimensions.Count < 2 Then
      MsgBox "You must select at least 2 dimensions for this operation."
      Exit Sub
   End If

   ' Ask the user if he/she wants vertical or horizontal alignment.
   Dim bHorizontal As Boolean
   If MsgBox("Do you want horizontal alignment? (Selecting ""No"" results in vertical alignment)", vbQuestion + vbYesNo, "Align Dimens
      bHorizontal = True
   Else
    bHorizontal = False
   End If

   For i = 1 To colDimensions.Count
      Dim oDimension As DrawingDimension
      Set oDimension = colDimensions.Item(i)

      If i = 1 Then
         ' Get the position of the first dimension text. This is
         ' the position the other dimensions will be aligned to.
         Dim dPosition As Double
         If bHorizontal Then
            dPosition = oDimension.Text.Origin.Y
         Else
            dPosition = oDimension.Text.Origin.X
         End If
      Else
         ' Change the position of the dimension.
         Dim oPosition As Point2d
         Set oPosition = oDimension.Text.Origin
         If bHorizontal Then
            oPosition.Y = dPosition
         Else
            oPosition.X = dPosition
         End If

         oDimension.Text.Origin = oPosition
      End If
   Next
End Sub
```

The first dimension selected defines the origin of the axis. A drawing document must be open and at least two dimensions selected.

Copy Code

```
    Dim oDrawDoc As DrawingDocument
   oDrawDoc = ThisApplication.ActiveDocument

   ' Determine if there are any dimensions in the select set.
   Dim oSelectSet As SelectSet
   oSelectSet = oDrawDoc.SelectSet
   Dim colDimensions As New Collection
   Dim i As Long
   For i = 1 To oSelectSet.Count
      If Typeof oSelectSet.Item(i) Is DrawingDimension Then
         ' Add any dimensions to the collection. We need to save them
         ' in something besides the selection set because once we start
         ' manipulating them the select set will be cleared.
```

```
        colDimensions.Add(oSelectSet.Item(i))
      End If
   Next

   If colDimensions.Count < 2 Then
      MsgBox("You must select at least 2 dimensions for this operation.")
      Exit Sub
   End If

   ' Ask the user if he/she wants vertical or horizontal alignment.
   Dim bHorizontal As Boolean
   If MsgBox("Do you want horizontal alignment? (Selecting ""No"" results in vertical alignment)", vbQuestion + vbYesNo, "Align Dimens
       bHorizontal = True
   Else
    bHorizontal = False
   End If

   Dim dPosition As Double
   For i = 1 To colDimensions.Count
      Dim oDimension As DrawingDimension
      oDimension = colDimensions.Item(i)

      If i = 1 Then
         ' Get the position of the first dimension text. This is
         ' the position the other dimensions will be aligned to.

         If bHorizontal Then
            dPosition = oDimension.Text.Origin.Y
         Else
            dPosition = oDimension.Text.Origin.X
         End If
      Else
         ' Change the position of the dimension.
         Dim oPosition As Point2d
         oPosition = oDimension.Text.Origin
         If bHorizontal Then
            oPosition.Y = dPosition
         Else
            oPosition.X = dPosition
         End If

         oDimension.Text.Origin = oPosition
      End If
   Next
```

Border Create and Insert

Description

This sample illustrates creating a new border definition object and using it for a sheet.

Code Samples

- VBA
- iLogic

This sample consists of two sub procedures. The first creates the border definition. The second inserts it into the active sheet. To run the sample have a drawing document open and run the CreateBorderDefinition first. After that you can run the InsertBorderOnSheet.

Copy Code

```
Sub Main()
        CreateBorderDefinition()
        InsertCustomBorderOnSheet()
End Sub

Sub CreateBorderDefinition()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new borderdefinition.
    Dim oBorderDef As BorderDefinition
    oBorderDef = oDrawDoc.BorderDefinitions.Add("Sample Border")

    ' Open the border definition's sketch for edit.  This is done by calling the Edit
    ' method of the BorderDefinition to obtain a DrawingSketch.  This actually creates
    ' a copy of the border definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oBorderDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add geometry.
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(2, 2), oTG.CreatePoint2d(25.94, 19.59))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 10.795), oTG.CreatePoint2d(2, 10.795))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(13.97, 0), oTG.CreatePoint2d(13.97, 2))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(25.94, 10.795), oTG.CreatePoint2d(27.94, 10.795))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(13.97, 19.59), oTG.CreatePoint2d(13.97, 21.59))

    ' Add some text to the border.
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(2, 1), "Here is a sample string")
    oTextBox.VerticalJustification = kAlignTextMiddle

    ' Add some prompted text to the border.
```

```
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(2, 20.59), "Enter designers name:")
    oTextBox.VerticalJustification = kAlignTextMiddle

    Call oBorderDef.ExitEdit(True)
End Sub

Public Sub InsertCustomBorderOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired border definition.
    Dim oBorderDef As BorderDefinition
    oBorderDef = oDrawDoc.BorderDefinitions.Item("Sample Border")

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' This border definition contains one prompted string input.  An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(0 To 0) As String
    sPromptStrings(0) = "This is the input for the prompted text."

    ' Add an instance of the border definition to the sheet.
    Dim oBorder As Border
    oBorder = oSheet.AddBorder(oBorderDef, sPromptStrings)
End Sub
```

This sample consists of two sub procedures. The first creates the border definition. The second inserts it into the active sheet. To run the sample have a drawing document open and run the CreateBorderDefinition sub first. After that you can run the InsertBorderOnSheet sub.

Copy Code

```
Sub Main
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create the new borderdefinition.
    Dim oBorderDef As BorderDefinition
    oBorderDef = oDrawDoc.BorderDefinitions.Add("Sample Border")

    ' Open the border definition's sketch for edit.  This is done by calling the Edit
    ' method of the BorderDefinition to obtain a DrawingSketch.  This actually creates
    ' a copy of the border definition's and opens it for edit.
    Dim oSketch As DrawingSketch
    Call oBorderDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Use the functionality of the sketch to add geometry.
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(2, 2), oTG.CreatePoint2d(25.94, 19.59))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 10.795), oTG.CreatePoint2d(2, 10.795))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(13.97, 0), oTG.CreatePoint2d(13.97, 2))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(25.94, 10.795), oTG.CreatePoint2d(27.94, 10.795))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(13.97, 19.59), oTG.CreatePoint2d(13.97, 21.59))

    ' Add some text to the border.
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(2, 1), "Here is a sample string")
    oTextBox.VerticalJustification = kAlignTextMiddle

    ' Add some prompted text to the border.
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(2, 20.59), "Enter designers name:")
    oTextBox.VerticalJustification = kAlignTextMiddle

    Call oBorderDef.ExitEdit(True)
End Sub

Public Sub InsertCustomBorderOnSheet()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Obtain a reference to the desired border definition.
    Dim oBorderDef As BorderDefinition
    oBorderDef = oDrawDoc.BorderDefinitions.Item("Sample Border")

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' This border definition contains one prompted string input.  An array
    ' must be input that contains the strings for the prompted strings.
    Dim sPromptStrings(0 To 0) As String
    sPromptStrings(0) = "This is the input for the prompted text."

    ' Add an instance of the border definition to the sheet.
    Dim oBorder As Border
    oBorder = oSheet.AddBorder(oBorderDef, sPromptStrings)
End Sub
```

Printing - All Sheets in Drawing

Description

This sample demonstrates how to print all sheets in a drawing document and set some of the DrawingPrintManager properties.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PlotAllSheetsInDrawing()
    'Print all sheets in drawing document
    'Get the active document and check whether it's drawing document
    If ThisApplication.ActiveDocument.DocumentType = kDrawingDocumentObject Then

        Dim oDrgDoc As DrawingDocument
        Set oDrgDoc = ThisApplication.ActiveDocument

        ' Set reference to drawing print manager
        ' DrawingPrintManager has more options than PrintManager
        ' as it's specific to drawing document
        Dim oDrgPrintMgr As DrawingPrintManager
        Set oDrgPrintMgr = oDrgDoc.PrintManager
        ' Set the printer name
        ' comment this line to use default printer or assign another one
        oDrgPrintMgr.Printer = "HP LaserJet 4000 Series PCL 6"

        'Set the paper size , scale and orientation
        oDrgPrintMgr.ScaleMode = kPrintBestFitScale
        oDrgPrintMgr.PaperSize = kPaperSizeA4
        oDrgPrintMgr.PrintRange = kPrintAllSheets
        oDrgPrintMgr.Orientation = kLandscapeOrientation
        oDrgPrintMgr.SubmitPrint
    End If
End Sub
```

Copy Code

```
    'Print all sheets in drawing document
    'Get the active document and check whether it's drawing document
    If ThisApplication.ActiveDocument.DocumentType = kDrawingDocumentObject Then

        Dim oDrgDoc As DrawingDocument
        oDrgDoc = ThisApplication.ActiveDocument

        ' Set reference to drawing print manager
        ' DrawingPrintManager has more options than PrintManager
        ' as it's specific to drawing document
        Dim oDrgPrintMgr As DrawingPrintManager
        oDrgPrintMgr = oDrgDoc.PrintManager
        ' Set the printer name
        ' comment this line to use default printer or assign another one
        oDrgPrintMgr.Printer = "HP LaserJet 4000 Series PCL 6"

        'Set the paper size , scale and orientation
        oDrgPrintMgr.ScaleMode = kPrintBestFitScale
        oDrgPrintMgr.PaperSize = kPaperSizeA4
        oDrgPrintMgr.PrintRange = kPrintAllSheets
        oDrgPrintMgr.Orientation = kLandscapeOrientation
        oDrgPrintMgr.SubmitPrint
    End If
```

Drawing Sketch Hatch Region Sample

Description

This sample demonstrates how to create a sketch hatch region in drawing.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub DrawingSketchHatchRegionSample()
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oSketch As DrawingSketch
    Set oSketch = oDoc.ActiveSheet.Sketches.Add
    oSketch.Edit

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim results As SketchEntitiesEnumerator
    Set results = oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(15, 12))
```

```
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    Dim oHatchPattern As DrawingHatchPattern
    Set oHatchPattern = oDoc.DrawingHatchPatternsManager.Item("ANSI 31")

    Dim oHatchRegion As SketchHatchRegion
    Set oHatchRegion = oSketch.SketchHatchRegions.Add(oProfile, oHatchPattern)

    oSketch.ExitEdit
End Sub
```

Copy Code

```
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oSketch As DrawingSketch
    oSketch = oDoc.ActiveSheet.Sketches.Add
    oSketch.Edit

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim results As SketchEntitiesEnumerator
    results = oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(15, 12))


    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    Dim oHatchPattern As DrawingHatchPattern
    oHatchPattern = oDoc.DrawingHatchPatternsManager.Item("ANSI 31")

    Dim oHatchRegion As SketchHatchRegion
    oHatchRegion = oSketch.SketchHatchRegions.Add(oProfile, oHatchPattern)

    oSketch.ExitEdit
```

Adding and editing a feature control frame

Description

These samples demonstrate editing an existing feature control frame symbol. The first sample adds a row to an existing symbol. The second sample replaces all rows of an existing symbol.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub EditFeatureControlFrame()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Get the feature control frame selection
    Dim oFCF As FeatureControlFrame
    Set oFCF = oDrawDoc.SelectSet.Item(1)

    ' Add a new row to the FCF symbol
    Dim oNewRow As FeatureControlFrameRow
    Set oNewRow = oFCF.FeatureControlFrameRows.Add(kParallelism, "0.40", , "C")
End Sub

Public Sub BatchUpdateFeatureControlFrameRows()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Get the feature control frame selection
    Dim oFCF As FeatureControlFrame
    Set oFCF = oDrawDoc.SelectSet.Item(1)

    ' Create a FeatureControlFrameRows object to define the symbol's rows
    Dim oRows As FeatureControlFrameRows
    Set oRows = oActiveSheet.FeatureControlFrames.CreateFeatureControlFrameRows

    ' Add two rows
    Call oRows.Add(kProfileOfAnySurface, "0.20", , "A", "B")
    Call oRows.Add(kParallelism, "0.40", , "C")

    oFCF.FeatureControlFrameRows = oRows
End Sub
```

Copy Code

```
Sub Main
        EditFeatureControlFrame()
        'BatchUpdateFeatureControlFrameRows()
End Sub

Sub EditFeatureControlFrame()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Get the feature control frame selection
    Dim oFCF As FeatureControlFrame
    oFCF = oDrawDoc.SelectSet.Item(1)

    ' Add a new row to the FCF symbol
    Dim oNewRow As FeatureControlFrameRow
    oNewRow = oFCF.FeatureControlFrameRows.Add(kParallelism, "0.40", , "C")
End Sub

Public Sub BatchUpdateFeatureControlFrameRows()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Get the feature control frame selection
    Dim oFCF As FeatureControlFrame
    oFCF = oDrawDoc.SelectSet.Item(1)

    ' Create a FeatureControlFrameRows object to define the symbol's rows
    Dim oRows As FeatureControlFrameRows
    oRows = oActiveSheet.FeatureControlFrames.CreateFeatureControlFrameRows

    ' Add two rows
    Call oRows.Add(kProfileOfAnySurface, "0.20", , "A", "B")
    Call oRows.Add(kParallelism, "0.40", , "C")

    oFCF.FeatureControlFrameRows = oRows
End Sub
```

Border Insert

Description

This sample illustrates inserting a default border.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub InsertDefaultBorder()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' Define the values to use as input for the border creation.
    Dim HorizontalZoneCount As Long
    HorizontalZoneCount = 15

    Dim HorizontalZoneLabelMode As BorderLabelModeEnum
    HorizontalZoneLabelMode = kBorderLabelModeNumeric

    Dim VerticalZoneCount As Long
    VerticalZoneCount = 10

    Dim VerticalZoneLabelMode As BorderLabelModeEnum
    VerticalZoneLabelMode = kBorderLabelModeAlphabetical

    Dim LabelFromBottomRight As Boolean
    LabelFromBottomRight = False

    Dim DelimitByLines As Boolean
    DelimitByLines = False

    Dim CenterMarks As Boolean
    CenterMarks = False

    Dim TopMargin As Double
    TopMargin = 5

    Dim BottomMargin As Double
```

```
    BottomMargin = 3

    Dim LeftMargin As Double
    LeftMargin = 1

    Dim RightMargin As Double
    RightMargin = 2

    ' Add the border to the sheet.  This sets all of the values, but any of them
    ' can be left out and it will default to an appropriate value.
    Dim oBorder As DefaultBorder
    Set oBorder = oSheet.AddDefaultBorder(HorizontalZoneCount, HorizontalZoneLabelMode, _
                                          VerticalZoneCount, VerticalZoneLabelMode, _
                                          LabelFromBottomRight, DelimitByLines, _
                                          CenterMarks, TopMargin, BottomMargin, _
                                          LeftMargin, RightMargin)
End Sub
```

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' Define the values to use as input for the border creation.
    Dim HorizontalZoneCount As Long
    HorizontalZoneCount = 15

    Dim HorizontalZoneLabelMode As BorderLabelModeEnum
    HorizontalZoneLabelMode = kBorderLabelModeNumeric

    Dim VerticalZoneCount As Long
    VerticalZoneCount = 10

    Dim VerticalZoneLabelMode As BorderLabelModeEnum
    VerticalZoneLabelMode = kBorderLabelModeAlphabetical

    Dim LabelFromBottomRight As Boolean
    LabelFromBottomRight = False

    Dim DelimitByLines As Boolean
    DelimitByLines = False

    Dim CenterMarks As Boolean
    CenterMarks = False

    Dim TopMargin As Double
    TopMargin = 5

    Dim BottomMargin As Double
    BottomMargin = 3

    Dim LeftMargin As Double
    LeftMargin = 1

    Dim RightMargin As Double
    RightMargin = 2

    ' Add the border to the sheet.  This sets all of the values, but any of them
    ' can be left out and it will default to an appropriate value.
    Dim oBorder As DefaultBorder
    oBorder = oSheet.AddDefaultBorder(HorizontalZoneCount, HorizontalZoneLabelMode, _
                                          VerticalZoneCount, VerticalZoneLabelMode, _
                                          LabelFromBottomRight, DelimitByLines, _
                                          CenterMarks, TopMargin, BottomMargin, _
                                          LeftMargin, RightMargin)
```

Border Insert Default

Description

This sample illustrates inserting a default border using the default values.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub InsertDefaultBorderUsingDefaults()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet
```

```
    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' Add the border to the sheet.
    Dim oBorder As DefaultBorder
    Set oBorder = oSheet.AddDefaultBorder()
End Sub
```

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to see if the sheet already has a border and delete it if it does.
    If Not oSheet.Border Is Nothing Then
        oSheet.Border.Delete
    End If

    ' Add the border to the sheet.
    Dim oBorder As DefaultBorder
    oBorder = oSheet.AddDefaultBorder()
```

Sketched Symbol Definition Library Creation

Description

This sample demonstrates how to create a sketched symbol definition and save it to the SketchedSymbolDefinitionLibrary, and then add the sketched symbol definition from the SketchedSymbolDefinitionLibrary to another drawing document.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub SketchSymbolDefinitionLibrarySample()
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oSymbolLib As SketchedSymbolDefinitionLibrary
    Dim oSymbolLibs As SketchedSymbolDefinitionLibraries

    Set oSymbolLibs = oDoc.SketchedSymbolDefinitions.SketchedSymbolDefinitionLibraries

    On Error Resume Next
    Set oSymbolLib = oSymbolLibs.Item("MySymbolLib")

    If oSymbolLib Is Nothing Then
        Set oSymbolLib = oSymbolLibs.Add("MySymbolLib")
    End If

    On Error GoTo 0

    ' Create a new sketched symbol definition.
    Dim oSymbolDef As SketchedSymbolDefinition
    Set oSymbolDef = oDoc.SketchedSymbolDefinitions.Add("CircleSymbol")

    Dim oSketch As DrawingSketch
    Call oSymbolDef.Edit(oSketch)

    Dim oCenter As Point2d
    Set oCenter = ThisApplication.TransientGeometry.CreatePoint2d(12, 12)

    Call oSketch.SketchCircles.AddByCenterRadius(oCenter, 0.5)

    Call oSymbolDef.ExitEdit(True)

    ' Save the above sketched symbol definition to library.
    Call oSymbolDef.SaveToLibrary(oSymbolLib, , , True)

    ' Sync the sketched symbol definition from the library to another drawing document
    Dim oNewDoc As DrawingDocument
    Set oNewDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oNewSymbolDef As SketchedSymbolDefinition
    Set oNewSymbolDef = oNewDoc.SketchedSymbolDefinitions.AddFromLibrary(oSymbolLib, "CircleSymbol")

    ' Create a sketched symbol bases on the synced sketched symbol definition from library.
    Call oNewDoc.Sheets(1).SketchedSymbols.Add(oNewSymbolDef, ThisApplication.TransientGeometry.CreatePoint2d(12, 12))
End Sub
```

Copy Code

```
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oSymbolLib As SketchedSymbolDefinitionLibrary
```

```
    Dim oSymbolLibs As SketchedSymbolDefinitionLibraries

    oSymbolLibs = oDoc.SketchedSymbolDefinitions.SketchedSymbolDefinitionLibraries

    On Error Resume Next
    oSymbolLib = oSymbolLibs.Item("MySymbolLib")

    If oSymbolLib Is Nothing Then
        oSymbolLib = oSymbolLibs.Add("MySymbolLib")
    End If

    On Error GoTo 0

    ' Create a new sketched symbol definition.
    Dim oSymbolDef As SketchedSymbolDefinition
    oSymbolDef = oDoc.SketchedSymbolDefinitions.Add("CircleSymbol")

    Dim oSketch As DrawingSketch
    Call oSymbolDef.Edit(oSketch)

    Dim oCenter As Point2d
    oCenter = ThisApplication.TransientGeometry.CreatePoint2d(12, 12)

    Call oSketch.SketchCircles.AddByCenterRadius(oCenter, 0.5)

    Call oSymbolDef.ExitEdit(True)

    ' Save the above sketched symbol definition to library.
    Call oSymbolDef.SaveToLibrary(oSymbolLib, , , True)

    ' Sync the sketched symbol definition from the library to another drawing document
    Dim oNewDoc As DrawingDocument
    oNewDoc = ThisApplication.Documents.Add(kDrawingDocumentObject)

    Dim oNewSymbolDef As SketchedSymbolDefinition
    oNewSymbolDef = oNewDoc.SketchedSymbolDefinitions.AddFromLibrary(oSymbolLib, "CircleSymbol")

    ' Create a sketched symbol bases on the synced sketched symbol definition from library.
    Call oNewDoc.Sheets(1).SketchedSymbols.Add(oNewSymbolDef, ThisApplication.TransientGeometry.CreatePoint2d(12, 12))
```

Custom Table - create

Description

This sample demonstrates how to create a custom table.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateCustomTable()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Set the column titles
    Dim oTitles(1 To 3) As String
    oTitles(1) = "Part Number"
    oTitles(2) = "Quantity"
    oTitles(3) = "Material"

    ' Set the contents of the custom table (contents are set row-wise)
    Dim oContents(1 To 9) As String
    oContents(1) = "1"
    oContents(2) = "1"
    oContents(3) = "Brass"
    oContents(4) = "2"
    oContents(5) = "2"
    oContents(6) = "Aluminium"
    oContents(7) = "3"
    oContents(8) = "1"
    oContents(9) = "Steel"

    ' Set the column widths (defaults to the column title width if not specified)
    Dim oColumnWidths(1 To 3) As Double
    oColumnWidths(1) = 2.5
    oColumnWidths(2) = 2.5
    oColumnWidths(3) = 4

    ' Create the custom table
    Dim oCustomTable As CustomTable
    Set oCustomTable = oSheet.CustomTables.Add("My Table", ThisApplication.TransientGeometry.CreatePoint2d(15, 15), _
                                       3, 3, oTitles, oContents, oColumnWidths)

    ' Change the 3rd column to be left justified.
    oCustomTable.Columns.Item(3).ValueHorizontalJustification = kAlignTextLeft

    ' Create a table format object
    Dim oFormat As TableFormat
    Set oFormat = oSheet.CustomTables.CreateTableFormat
```

```
    ' Set inside line color to red.
    oFormat.InsideLineColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0)

    ' Set outside line weight.
    oFormat.OutsideLineWeight = 0.1

    ' Modify the table formats
    oCustomTable.OverrideFormat = oFormat
End Sub
```

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Set the column titles
    Dim oTitles(0 To 2) As String
    oTitles(0) = "Part Number"
    oTitles(1) = "Quantity"
    oTitles(2) = "Material"

    ' Set the contents of the custom table (contents are set row-wise)
    Dim oContents(0 To 8) As String
    oContents(0) = "1"
    oContents(1) = "1"
    oContents(2) = "Brass"
    oContents(3) = "2"
    oContents(4) = "2"
    oContents(5) = "Aluminium"
    oContents(6) = "3"
    oContents(7) = "1"
    oContents(8) = "Steel"

    ' Set the column widths (defaults to the column title width if not specified)
    Dim oColumnWidths(0 To 2) As Double
    oColumnWidths(0) = 2.5
    oColumnWidths(1) = 2.5
    oColumnWidths(2) = 4

    ' Create the custom table
    Dim oCustomTable As CustomTable
    oCustomTable = oSheet.CustomTables.Add("My Table", ThisApplication.TransientGeometry.CreatePoint2d(15, 15), _
                                           3, 3, oTitles, oContents, oColumnWidths)

    ' Change the 3rd column to be left justified.
    oCustomTable.Columns.Item(3).ValueHorizontalJustification = kAlignTextLeft

    ' Create a table format object
    Dim oFormat As TableFormat
    oFormat = oSheet.CustomTables.CreateTableFormat

    ' Set inside line color to red.
    oFormat.InsideLineColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0)

    ' Set outside line weight.
    oFormat.OutsideLineWeight = 0.1

    ' Modify the table formats
    oCustomTable.OverrideFormat = oFormat
```

Create a Bend Table

Description

This sample demonstrates the creation of a bend table in a drawing from a sheet metal part.

Code Samples

- VBA
- iLogic

Change the location of the source part in the code below before running the sample.

Copy Code

```
Public Sub CreateBendTable()
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Specify the columns - this can be specified in any combination/order
    Dim strColumns(1 To 5) As String
    strColumns(1) = "BEND ID"
```

```
    strColumns(2) = "BEND RADIUS"
    strColumns(3) = "BEND ANGLE"
    strColumns(4) = "BEND KFACTOR"
    strColumns(5) = "BEND DIRECTION"

    ' Create the bend table by specifying the source sheet metal file
    Dim oBendTable As CustomTable
    Set oBendTable = oActiveSheet.CustomTables.AddBendTable("C:\Temp\SheetMetal.ipt", oPoint, "Bend Table", strColumns)
End Sub
```

Change the location of the source part in the code below before running the sample.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Specify the columns - this can be specified in any combination/order
    Dim strColumns(0 To 4) As String
    strColumns(0) = "BEND ID"
    strColumns(1) = "BEND RADIUS"
    strColumns(2) = "BEND ANGLE"
    strColumns(3) = "BEND KFACTOR"
    strColumns(4) = "BEND DIRECTION"

    ' Create the bend table by specifying the source sheet metal file
    Dim oBendTable As CustomTable
    oBendTable = oActiveSheet.CustomTables.AddBendTable("C:\Temp\SheetMetal.ipt", oPoint, "Bend Table", strColumns)
```

Create a Configuration Table

Description

This sample demonstrates the creation of a configuration (iPart/iAssembly) table in a drawing from a factory document.

Code Samples

- VBA
- iLogic

Change the location of the source part in the code below before running the sample.

Copy Code

```
Public Sub CreateConfigurationTable()
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Open the factory document invisible
    Dim oiPartDoc As PartDocument
    Set oiPartDoc = ThisApplication.Documents.Open("C:\Temp\iPartFactory.ipt", False)

    Dim oiPartFactory As iPartFactory
    Set oiPartFactory = oiPartDoc.ComponentDefinition.iPartFactory

    ' Specify the columns - this sample includes all columns from the factory table
    Dim strColumns() As String
    Redim strColumns(oiPartFactory.TableColumns.Count - 1)

    Dim i As Long
    For i = 1 To oiPartFactory.TableColumns.Count
        strColumns(i - 1) = oiPartFactory.TableColumns.Item(i).Heading
    Next

    ' Release reference on factory document since we opened it invisible
    oiPartDoc.ReleaseReference

    ' Create the configuration table by specifying the source iPart/iAssembly file
    Dim oConfigTable As CustomTable
    Set oConfigTable = oActiveSheet.CustomTables.AddConfigurationTable("C:\Temp\iPartFactory.ipt", oPoint, "Configuration Table", strC
End Sub
```

Change the location of the source part in the code below before running the sample.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
```

```
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Open the factory document invisible
    Dim oiPartDoc As PartDocument
    oiPartDoc = ThisApplication.Documents.Open("C:\Temp\iPartFactory.ipt", False)

    Dim oiPartFactory As iPartFactory
    oiPartFactory = oiPartDoc.ComponentDefinition.iPartFactory

    ' Specify the columns - this sample includes all columns from the factory table
    Dim strColumns() As String
    Redim strColumns(oiPartFactory.TableColumns.Count - 1)

    Dim i As Long
    For i = 1 To oiPartFactory.TableColumns.Count
        strColumns(i - 1) = oiPartFactory.TableColumns.Item(i).Heading
    Next

    ' Release reference on factory document since we opened it invisible
    oiPartDoc.ReleaseReference

    ' Create the configuration table by specifying the source iPart/iAssembly file
    Dim oConfigTable As CustomTable
    oConfigTable = oActiveSheet.CustomTables.AddConfigurationTable("C:\Temp\iPartFactory.ipt", oPoint, "Configuration Table", strColum
```

Create a drawing Excel Table

Description

This sample demonstrates the creation of a table based on an Excel file in a drawing.

Code Samples

- VBA
- iLogic

Change the location of the source Excel file in the code below before running the sample.

Copy Code

```
Public Sub CreateDrawingExcelTable()
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Create the table by specifying the source Excel file
    ' This assumes that the first row in the Excel sheet specifies the column headers
    ' You can specify a different row using the last argument of the AddExcelTable method
    Dim oExcelTable As CustomTable
    Set oExcelTable = oActiveSheet.CustomTables.AddExcelTable("C:\Temp\test.xls", oPoint, "Excel Table")
End Sub
```

Change the location of the source Excel file in the code below before running the sample.

Copy Code

```
    ' Set a reference to the active drawing document
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Create the placement point for the table
    Dim oPoint As Point2d
    oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Create the table by specifying the source Excel file
    ' This assumes that the first row in the Excel sheet specifies the column headers
    ' You can specify a different row using the last argument of the AddExcelTable method
    Dim oExcelTable As CustomTable
    oExcelTable = oActiveSheet.CustomTables.AddExcelTable("C:\Temp\test.xls", oPoint, "Excel Table")
```

Creating hole tables

Description

This sample demonstrates the creation of hole tables in a drawing.

Code Samples

- VBA
- iLogic

Select a drawing view that contains holes and run the following sample.

Copy Code

```
Sub CreateHoleTables()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    Set oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing view.
    ' This assumes that a drawing view is selected.
    Dim oDrawingView As DrawingView
    Set oDrawingView = oDrawDoc.SelectSet.Item(1)

    ' Create origin indicator if it has not been already created.
    If Not oDrawingView.HasOriginIndicator Then
        ' Create point intent to anchor the origin to.
        Dim oDimIntent As GeometryIntent
        Dim oPointIntent As Point2d

        ' Get the first curve on the view
        Dim oCurve As DrawingCurve
        Set oCurve = oDrawingView.DrawingCurves.Item(1)

        ' Check if it has a strt point
        Set oPointIntent = oCurve.StartPoint

        If oPointIntent Is Nothing Then
            ' Else use the center point
            Set oPointIntent = oCurve.CenterPoint
        End If

        Set oDimIntent = oActiveSheet.CreateGeometryIntent(oCurve, oPointIntent)

        oDrawingView.CreateOriginIndicator oDimIntent
    End If

    Dim oPlacementPoint As Point2d

    ' Set a reference to th sheet's border
    Dim oBorder As Border
    Set oBorder = oActiveSheet.Border

    If Not oBorder Is Nothing Then
        ' A border exists. The placement point
        ' is the top-left corner of the border.
        Set oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oBorder.RangeBox.MinPoint.X, oBorder.RangeBox.MaxPoint.Y
    Else
        ' There is no border. The placement point
        ' is the top-left corner of the sheet.
        Set oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(0, oActiveSheet.height)
    End If

    ' Create a 'view' hole table
    ' This hole table includes all holes as specified by the active hole table style
    Dim oViewHoleTable As HoleTable
    Set oViewHoleTable = oActiveSheet.HoleTables.Add(oDrawingView, oPlacementPoint)

    oPlacementPoint.X = oActiveSheet.width / 2

    ' Create a 'feature type' hole table
    ' This hole table includes specified hole types only
    Dim oFeatureHoleTable As HoleTable
    Set oFeatureHoleTable = oActiveSheet.HoleTables.AddByFeatureType(oDrawingView, oPlacementPoint, _
            True, True, True, True, False, False, False)
End Sub
```

Select a drawing view that contains holes and run the following sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active sheet.
    Dim oActiveSheet As Sheet
    oActiveSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the drawing view.
    ' This assumes that a drawing view is selected.
    Dim oDrawingView As DrawingView
    oDrawingView = oDrawDoc.SelectSet.Item(1)

    ' Create origin indicator if it has not been already created.
    If Not oDrawingView.HasOriginIndicator Then
        ' Create point intent to anchor the origin to.
        Dim oDimIntent As GeometryIntent
        Dim oPointIntent As Point2d

        ' Get the first curve on the view
        Dim oCurve As DrawingCurve
        oCurve = oDrawingView.DrawingCurves.Item(1)

        ' Check if it has a strt point
```

```
        oPointIntent = oCurve.StartPoint

        If oPointIntent Is Nothing Then
            ' Else use the center point
            oPointIntent = oCurve.CenterPoint
        End If

        oDimIntent = oActiveSheet.CreateGeometryIntent(oCurve, oPointIntent)

        oDrawingView.CreateOriginIndicator(oDimIntent)
    End If

    Dim oPlacementPoint As Point2d

    ' Set a reference to th sheet's border
    Dim oBorder As Border
    oBorder = oActiveSheet.Border

    If Not oBorder Is Nothing Then
        ' A border exists. The placement point
        ' is the top-left corner of the border.
        oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oBorder.RangeBox.MinPoint.X, oBorder.RangeBox.MaxPoint.Y)
    Else
        ' There is no border. The placement point
        ' is the top-left corner of the sheet.
        oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(0, oActiveSheet.height)
    End If

    ' Create a 'view' hole table
    ' This hole table includes all holes as specified by the active hole table style
    Dim oViewHoleTable As HoleTable
    oViewHoleTable = oActiveSheet.HoleTables.Add(oDrawingView, oPlacementPoint)

    oPlacementPoint.X = oActiveSheet.width / 2

    ' Create a 'feature type' hole table
    ' This hole table includes specified hole types only
    Dim oFeatureHoleTable As HoleTable
    oFeatureHoleTable = oActiveSheet.HoleTables.AddByFeatureType(oDrawingView, oPlacementPoint, _
            True, True, True, True, False, False, False)
```

Creating a parts list

Description

This sample demonstrates the creation of a parts list. The parts list is placed at the top right corner of the border if one exists, else it is placed at the top right corner of the sheet.

Code Samples

- VBA
- iLogic

To run this sample, have a drawing document open. The active sheet in the drawing should have at least one drawing view and the first drawing view on the sheet should not be a draft view.

Copy Code

```
Public Sub CreatePartsList()
    On Error Resume Next

    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the first drawing view on
    ' the sheet. This assumes the first drawing
    ' view on the sheet is not a draft view.
    Dim oDrawingView As DrawingView
    Set oDrawingView = oSheet.DrawingViews(1)

    ' Set a reference to th sheet's border
    Dim oBorder As Border
    Set oBorder = oSheet.Border

    Dim oPlacementPoint As Point2d

    If Not oBorder Is Nothing Then
        ' A border exists. The placement point
        ' is the top-right corner of the border.
        Set oPlacementPoint = oBorder.RangeBox.MaxPoint
    Else
        ' There is no border. The placement point
        ' is the top-right corner of the sheet.
        Set oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oSheet.Width, oSheet.height)
    End If

    ' Create the parts list.
    Dim oPartsList As PartsList
    Set oPartsList = oSheet.PartsLists.Add(oDrawingView, oPlacementPoint)
End Sub
```

To run this sample, have a drawing document open. The active sheet in the drawing should have at least one drawing view and the first drawing view on the sheet should not be a draft view.

Copy Code

```
    On Error Resume Next

    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Set a reference to the first drawing view on
    ' the sheet. This assumes the first drawing
    ' view on the sheet is not a draft view.
    Dim oDrawingView As DrawingView
    oDrawingView = oSheet.DrawingViews(1)

    ' Set a reference to th sheet's border
    Dim oBorder As Border
    oBorder = oSheet.Border

    Dim oPlacementPoint As Point2d

    If Not oBorder Is Nothing Then
        ' A border exists. The placement point
        ' is the top-right corner of the border.
        oPlacementPoint = oBorder.RangeBox.MaxPoint
    Else
        ' There is no border. The placement point
        ' is the top-right corner of the sheet.
        oPlacementPoint = ThisApplication.TransientGeometry.CreatePoint2d(oSheet.Width, oSheet.height)
    End If

    ' Create the parts list.
    Dim oPartsList As PartsList
    oPartsList = oSheet.PartsLists.Add(oDrawingView, oPlacementPoint)
```

Parts List Edit

Description

This sample illustrates editing the contents of the parts list.

Code Samples

- VBA
- iLogic

To run this sample have a sheet active that contains a parts list.

Copy Code

```
Public Sub PartListEdit()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first parts list on the active sheet.
    ' This assumes that a parts list is on the active sheet.
    Dim oPartList As PartsList
    Set oPartList = oDrawDoc.ActiveSheet.PartsLists.Item(1)

    ' Iterate through the contents of the parts list.
    Dim i As Long
    For i = 1 To oPartList.PartsListRows.Count
        ' Get the current row.
        Dim oRow As PartsListRow
        Set oRow = oPartList.PartsListRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oPartList.PartsListColumns.Count
            ' Get the current cell.
            Dim oCell As PartsListCell
            Set oCell = oRow.Item(j)

            ' Check that the column isn't the quantity column.
            If oPartList.PartsListColumns.Item(j).Title <> "QTY" Then
                ' Change the current value in the part list.
                oCell.Value = i & "," & j
            End If
        Next
    Next

    ' This changes a specific column by name.
    Dim ItemNumber As Long
    ItemNumber = oPartList.PartsListRows.Count
    For i = 1 To oPartList.PartsListRows.Count
        Set oCell = oPartList.PartsListRows.Item(i).Item("ITEM")
        oCell.Value = ItemNumber
        ItemNumber = ItemNumber - 1
```

```
    Next
End Sub
```

To run this sample have a sheet active that contains a parts list.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first parts list on the active sheet.
    ' This assumes that a parts list is on the active sheet.
    Dim oPartList As PartsList
    oPartList = oDrawDoc.ActiveSheet.PartsLists.Item(1)

    ' Iterate through the contents of the parts list.
    Dim i As Long
    Dim oCell As PartsListCell

    For i = 1 To oPartList.PartsListRows.Count
        ' Get the current row.
        Dim oRow As PartsListRow
        oRow = oPartList.PartsListRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oPartList.PartsListColumns.Count
            ' Get the current cell.

            oCell = oRow.Item(j)

            ' Check that the column isn't the quantity column.
            If oPartList.PartsListColumns.Item(j).Title <> "QTY" Then
                ' Change the current value in the part list.
                oCell.Value = i & "," & j
            End If
        Next
    Next

    ' This changes a specific column by name.
    Dim ItemNumber As Long
    ItemNumber = oPartList.PartsListRows.Count
    For i = 1 To oPartList.PartsListRows.Count
        oCell = oPartList.PartsListRows.Item(i).Item("ITEM")
        oCell.Value = ItemNumber
        ItemNumber = ItemNumber - 1
    Next
```

Parts List Query

Description

This sample illustrates querying the contents of the parts list.

Code Samples

- VBA
- iLogic

To run this sample have a sheet active that contains a parts list.

Copy Code

```
Public Sub PartListQuery()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first parts list on the active sheet.
    ' This assumes that a parts list is on the active sheet.
    Dim oPartList As PartsList
    Set oPartList = oDrawDoc.ActiveSheet.PartsLists.Item(1)

    ' Iterate through the contents of the parts list.
    Dim i As Long
    For i = 1 To oPartList.PartsListRows.Count
        ' Get the current row.
        Dim oRow As PartsListRow
        Set oRow = oPartList.PartsListRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oPartList.PartsListColumns.Count
            ' Get the current cell.
            Dim oCell As PartsListCell
            Set oCell = oRow.Item(j)

            ' Display the value of the current cell.
            Debug.Print "Row: " & i & ", Column: " & oPartList.PartsListColumns.Item(j).Title & " = "; oCell.Value
        Next
    Next
End Sub
```

To run this sample have a sheet active that contains a parts list.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first parts list on the active sheet.
    ' This assumes that a parts list is on the active sheet.
    Dim oPartList As PartsList
    oPartList = oDrawDoc.ActiveSheet.PartsLists.Item(1)

    ' Iterate through the contents of the parts list.
    Dim i As Long
    For i = 1 To oPartList.PartsListRows.Count
        ' Get the current row.
        Dim oRow As PartsListRow
        oRow = oPartList.PartsListRows.Item(i)

        ' Iterate through each column in the row.
        Dim j As Long
        For j = 1 To oPartList.PartsListColumns.Count
            ' Get the current cell.
            Dim oCell As PartsListCell
            oCell = oRow.Item(j)

            ' Display the value of the current cell.
            Logger.Info("Row: " & i & ", Column: " & oPartList.PartsListColumns.Item(j).Title & " = " & oCell.Value)

        Next
    Next
```

Creation of a break operation in a drawing view

Description

Demonstrates the creation of a break operation.

Code Samples

- VBA
- iLogic

Before running this sample, select a drawing view in the active drawing.

Copy Code

```
Public Sub CreateBreakoperationInDrawingView()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Check to make sure a drawing view is selected.
    If Not TypeOf oDrawDoc.SelectSet.Item(1) Is DrawingView Then
        MsgBox "A drawing view must be selected."
        Exit Sub
    End If

    ' Set a reference to the selected drawing. This assumes
    ' that the selected view is not a draft view.
    Dim oDrawingView As DrawingView
    Set oDrawingView = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the center of the base view.
    Dim oCenter As Point2d
    Set oCenter = oDrawingView.Center

    ' Define the start point of the break
    Dim oStartPoint As Point2d
    Set oStartPoint = ThisApplication.TransientGeometry.CreatePoint2d(oCenter.X - 1, oCenter.Y)

    ' Define the end point of the break
    Dim oEndPoint As Point2d
    Set oEndPoint = ThisApplication.TransientGeometry.CreatePoint2d(oCenter.X + 1, oCenter.Y)

    Dim oBreakOperation As BreakOperation
    Set oBreakOperation = oDrawingView.BreakOperations.Add(kHorizontalBreakOrientation, oStartPoint, oEndPoint, kRectangularBreakStyle
End Sub
```

Before running this sample, select a drawing view in the active drawing.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Check to make sure a drawing view is selected.
    If Not TypeOf oDrawDoc.SelectSet.Item(1) Is DrawingView Then
        MsgBox("A drawing view must be selected.")
        Exit Sub
```

```
    End If

    ' Set a reference to the selected drawing. This assumes
    ' that the selected view is not a draft view.
    Dim oDrawingView As DrawingView
    oDrawingView = oDrawDoc.SelectSet.Item(1)

    ' Set a reference to the center of the base view.
    Dim oCenter As Point2d
    oCenter = oDrawingView.Center

    ' Define the start point of the break
    Dim oStartPoint As Point2d
    oStartPoint = ThisApplication.TransientGeometry.CreatePoint2d(oCenter.X - 1, oCenter.Y)

    ' Define the end point of the break
    Dim oEndPoint As Point2d
    oEndPoint = ThisApplication.TransientGeometry.CreatePoint2d(oCenter.X + 1, oCenter.Y)

    Dim oBreakOperation As BreakOperation
    oBreakOperation = oDrawingView.BreakOperations.Add(kHorizontalBreakOrientation, oStartPoint, oEndPoint, kRectangularBreakStyle, 5)
```

Create BreakOpertion by Sketch Sample

Description

This sample demonstrates how to create a break operation using a sketch.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a break operation using a sketch. You need to open a drawing document with a drawing view on the active sheet before running this sample.

Copy Code

```
Sub CreateBreakOpertionBySketchSample()
    ' Open a drawing document and place a drawing view on the active sheet before running this sample.
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oView As DrawingView
    Set oView = oDoc.ActiveSheet.DrawingViews(1)

    Dim oSk As DrawingSketch
    Set oSk = oView.Sketches.Add
    oSk.Edit

    Dim oLine As SketchLine
    Dim oPt1 As Point2d, oPt2 As Point2d
    Set oPt1 = ThisApplication.TransientGeometry.CreatePoint2d(-oView.Width / 16, oView.Height / 2)
    Set oPt2 = ThisApplication.TransientGeometry.CreatePoint2d(-oView.Width / 16, -oView.Height / 2)

    Set oLine = oSk.SketchLines.AddByTwoPoints(oPt1, oPt2)

    Set oPt1 = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 16, oView.Height / 2)
    Set oPt2 = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 16, -oView.Height / 2)
    Set oLine = oSk.SketchLines.AddByTwoPoints(oPt1, oPt2)
    oSk.ExitEdit

    ' Create break operation bases on sketch
    Dim oBreak As BreakOperation
    Set oBreak = oView.BreakOperations.AddBySketch(oSk, kRectangularBreakStyle)
End Sub
```

This sample demonstrates how to create a break operation using a sketch. You need to open a drawing document with a drawing view on the active sheet before running this sample.

Copy Code

```
    ' Open a drawing document and place a drawing view on the active sheet before running this sample.
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oView As DrawingView
    oView = oDoc.ActiveSheet.DrawingViews(1)

    Dim oSk As DrawingSketch
    oSk = oView.Sketches.Add
    oSk.Edit

    Dim oLine As SketchLine
    Dim oPt1 As Point2d, oPt2 As Point2d
    oPt1 = ThisApplication.TransientGeometry.CreatePoint2d(-oView.Width / 16, oView.Height / 2)
    oPt2 = ThisApplication.TransientGeometry.CreatePoint2d(-oView.Width / 16, -oView.Height / 2)

    oLine = oSk.SketchLines.AddByTwoPoints(oPt1, oPt2)

    oPt1 = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 16, oView.Height / 2)
    oPt2 = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 16, -oView.Height / 2)
    oLine = oSk.SketchLines.AddByTwoPoints(oPt1, oPt2)
    oSk.ExitEdit

    ' Create break operation bases on sketch
    Dim oBreak As BreakOperation
    oBreak = oView.BreakOperations.AddBySketch(oSk, kRectangularBreakStyle)
```

CropOperation creation sample

Description

This sample demonstrates how to create a crop operation for a drawing view bases on a sketch under the drawing view.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a crop operation for a drawing view bases on a sketch under the drawing view. You need to open a drawing document with a drawing view in the active sheet before running this sample.

Copy Code

```
Sub CropOperationSample()
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    Set oSheet = oDoc.ActiveSheet

    Dim oView As DrawingView
    Set oView = oSheet.DrawingViews(1)

    Dim oSk As DrawingSketch
    Set oSk = oView.Sketches.Add

    oSk.Edit

    Dim oPtOne As Point2d
    Dim oPtTwo As Point2d
    Set oPtOne = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)
    Set oPtTwo = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 2, oView.Height / 2)

    ' Create a rectangler used for crop operation.
    oSk.SketchLines.AddAsTwoPointRectangle oPtOne, oPtTwo

    oSk.ExitEdit

    ' Create a CropOperation.
    Dim oCrop As CropOperation
    Set oCrop = oView.CropOperations.AddBySketch(oSk, True)
End Sub
```

This sample demonstrates how to create a crop operation for a drawing view bases on a sketch under the drawing view. You need to open a drawing document with a drawing view in the active sheet before running this sample.

Copy Code

```
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oSheet As Sheet
    oSheet = oDoc.ActiveSheet

    Dim oView As DrawingView
    oView = oSheet.DrawingViews(1)

    Dim oSk As DrawingSketch
    oSk = oView.Sketches.Add

    oSk.Edit

    Dim oPtOne As Point2d
    Dim oPtTwo As Point2d
    oPtOne = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)
    oPtTwo = ThisApplication.TransientGeometry.CreatePoint2d(oView.Width / 2, oView.Height / 2)

    ' Create a rectangler used for crop operation.
    oSk.SketchLines.AddAsTwoPointRectangle(oPtOne, oPtTwo)

    oSk.ExitEdit

    ' Create a CropOperation.
    Dim oCrop As CropOperation
    oCrop = oView.CropOperations.AddBySketch(oSk, True)
```

Drawing Sketches - editing line type and color

Description

This sample demonstrates the modification of sketch entity line type and color in drawings.

Code Samples

- VBA
- iLogic

Open a drawing document and run the following sample.

Copy Code

```
Public Sub ModifyDrawingSketchEntityProperties()
    ' Set a reference to the active document.  This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the drawing view on the active sheet.
    Dim oDrawView As DrawingView
    Set oDrawView = oDoc.ActiveSheet.DrawingViews.AddDraftView

    ' Set a reference to the sketch of the draft view.
    Dim oSketch As DrawingSketch
    Set oSketch = oDrawView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(30, 30))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 30), oTG.CreatePoint2d(30, 10))

    ' Create a transient color object.
    Dim oColor As Color
    Set oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0) 'Red

    ' Override the color of the first line.
    oSketch.SketchLines(1).OverrideColor = oColor

    ' Override the line type of the second line.
    oSketch.SketchLines(2).LineType = kDashedLineType

    ' Override the line weight of the second line.
    oSketch.SketchLines(2).LineWeight = 0.11

    ' Override the line scale of the second line.
    oSketch.SketchLines(2).LineScale = 0.5

    ' Exit from editing the sketch.
    oSketch.ExitEdit
End Sub
```

Open a drawing document and run the following sample.

Copy Code

```
    ' Set a reference to the active document.  This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the drawing view on the active sheet.
    Dim oDrawView As DrawingView
    oDrawView = oDoc.ActiveSheet.DrawingViews.AddDraftView

    ' Set a reference to the sketch of the draft view.
    Dim oSketch As DrawingSketch
    oSketch = oDrawView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(30, 30))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 30), oTG.CreatePoint2d(30, 10))

    ' Create a transient color object.
    Dim oColor As Color
    oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0) 'Red

    ' Override the color of the first line.
    oSketch.SketchLines(1).OverrideColor = oColor

    ' Override the line type of the second line.
    oSketch.SketchLines(2).LineType = kDashedLineType

    ' Override the line weight of the second line.
    oSketch.SketchLines(2).LineWeight = 0.11

    ' Override the line scale of the second line.
    oSketch.SketchLines(2).LineScale = 0.5

    ' Exit from editing the sketch.
    oSketch.ExitEdit
```

Sketch fill region

Description

This sample demonstrates the sketch fill functionality in drawing sketches.

Code Samples

- VBA

- iLogic

Have a drawing document open and run the sample.

Copy Code

```
Public Sub DrawingSketchFill()
    ' Set a reference to the active document. This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Create a sketch on the active sheet
    Dim oSketch As DrawingSketch
    Set oSketch = oDoc.ActiveSheet.Sketches.Add

    ' Put the sketch in edit mode
    oSketch.Edit

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Draw a circle in the sketch.
    Dim oCircle1 As SketchCircle
    Set oCircle1 = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(10, 30), 2)

    ' Create a collection and add the circle.
    Dim oCollection1 As ObjectCollection
    Set oCollection1 = ThisApplication.TransientObjects.CreateObjectCollection
    oCollection1.Add oCircle1

    ' Create a profile from the first circle
    Dim oProfile1 As Profile
    Set oProfile1 = oSketch.Profiles.AddForSolid(False, oCollection1)

    ' Create a fill region based on the layer color.
    Call oSketch.SketchFillRegions.Add(oProfile1)

    ' Draw another circle in the sketch.
    Dim oCircle2 As SketchCircle
    Set oCircle2 = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(30, 30), 2)

    ' Create a collection and add the circle.
    Dim oCollection2 As ObjectCollection
    Set oCollection2 = ThisApplication.TransientObjects.CreateObjectCollection
    oCollection2.Add oCircle2

    ' Create a profile from the second circle
    Dim oProfile2 As Profile
    Set oProfile2 = oSketch.Profiles.AddForSolid(False, oCollection2)

    ' Create a transient color object.
    Dim oColor As Color
    Set oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0) 'Red

    ' Create a fill region with an override color.
    Call oSketch.SketchFillRegions.Add(oProfile2, oColor)

    ' Exit from editing the sketch.
    oSketch.ExitEdit
End Sub
```

Have a drawing document open and run the sample.

Copy Code

```
    ' Set a reference to the active document. This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Create a sketch on the active sheet
    Dim oSketch As DrawingSketch
    oSketch = oDoc.ActiveSheet.Sketches.Add

    ' Put the sketch in edit mode
    oSketch.Edit

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Draw a circle in the sketch.
    Dim oCircle1 As SketchCircle
    oCircle1 = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(10, 30), 2)

    ' Create a collection and add the circle.
    Dim oCollection1 As ObjectCollection
    oCollection1 = ThisApplication.TransientObjects.CreateObjectCollection
    oCollection1.Add(oCircle1)

    ' Create a profile from the first circle
    Dim oProfile1 As Profile
    oProfile1 = oSketch.Profiles.AddForSolid(False, oCollection1)

    ' Create a fill region based on the layer color.
    Call oSketch.SketchFillRegions.Add(oProfile1)

    ' Draw another circle in the sketch.
    Dim oCircle2 As SketchCircle
    oCircle2 = oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(30, 30), 2)

    ' Create a collection and add the circle.
    Dim oCollection2 As ObjectCollection
    oCollection2 = ThisApplication.TransientObjects.CreateObjectCollection
    oCollection2.Add(oCircle2)
```

```
    ' Create a profile from the second circle
    Dim oProfile2 As Profile
    oProfile2 = oSketch.Profiles.AddForSolid(False, oCollection2)

    ' Create a transient color object.
    Dim oColor As Color
    oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0) 'Red

    ' Create a fill region with an override color.
    Call oSketch.SketchFillRegions.Add(oProfile2, oColor)

    ' Exit from editing the sketch.
    oSketch.ExitEdit
```

Sketch within a drawing view

Description

This sample demonstrates creating a sketch within an existing drawing view.

Code Samples

- VBA
- iLogic

To run this sample have a drawing open where the active sheet contains at least one drawing view.

Copy Code

```
Private Sub CreateDrawingSketchInView()
    ' Set a reference to the active document.  This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first drawing view on the active sheet.  This
    ' assumes the active sheet contains at least one drawing view.
    Dim oDrawView As DrawingView
    Set oDrawView = oDoc.ActiveSheet.DrawingViews.Item(1)

    ' Create the new drawing sketch.
    Dim oSketch As DrawingSketch
    Set oSketch = oDrawView.Sketches.Add

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Open the sketch for edit.
    oSketch.Edit

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(3, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 3), oTG.CreatePoint2d(3, 0))

    ' Exit from editing the sketch.
    oSketch.ExitEdit
End Sub
```

To run this sample have a drawing open where the active sheet contains at least one drawing view.

Copy Code

```
    ' Set a reference to the active document.  This assumes it
    ' is a drawing document.
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the first drawing view on the active sheet.  This
    ' assumes the active sheet contains at least one drawing view.
    Dim oDrawView As DrawingView
    oDrawView = oDoc.ActiveSheet.DrawingViews.Item(1)

    ' Create the new drawing sketch.
    Dim oSketch As DrawingSketch
    oSketch = oDrawView.Sketches.Add

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Open the sketch for edit.
    oSketch.Edit

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(3, 3))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 3), oTG.CreatePoint2d(3, 0))

    ' Exit from editing the sketch.
    oSketch.ExitEdit
```

Adding Representation views

Description

This sample demonstrates how to create a base view by specifying various representations.

Code Samples

- VBA
- iLogic

Before running this sample, make sure that the file C:\TempReps.iam exists (or change the path in the sample). The file must contain a model state named MyModelState, a positional representation named MyPositionalRep and a design view representation named MyDesignViewRep.

Copy Code

```
Public Sub AddBaseViewWithRepresentations()
  ' Set a reference to the drawing document.
  ' This assumes a drawing document is active.
  Dim oDrawDoc As DrawingDocument
  Set oDrawDoc = ThisApplication.ActiveDocument

  'Set a reference to the active sheet.
  Dim oSheet As Sheet
  Set oSheet = oDrawDoc.ActiveSheet

  ' Create a new NameValueMap object
  Dim oBaseViewOptions As NameValueMap
  Set oBaseViewOptions = ThisApplication.TransientObjects.CreateNameValueMap

  ' Set the representations to use when creating the base view.
  Call oBaseViewOptions.Add("PositionalRepresentation", "MyPositionalRep")
  Call oBaseViewOptions.Add("DesignViewRepresentation", "MyDesignViewRep")
  Call oBaseViewOptions.Add("DesignViewAssociative", True)

  ' Open the model document (corresponding to the "MyModelState" model state).
  Dim strFullDocumentName As String
  strFullDocumentName = ThisApplication.FileManager.GetFullDocumentName("C:\tempreps.iam", "MyModelState")

  Dim oModel As Document
  Set oModel = ThisApplication.Documents.Open(strFullDocumentName, False)

  ' Create the placement point object.
  Dim oPoint As Point2d
  Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

  ' Create a base view.
  Dim oBaseView As DrawingView
  Set oBaseView = oSheet.DrawingViews.AddBaseView(oModel, oPoint, 2, _
  kIsoTopLeftViewOrientation, kHiddenLineRemovedDrawingViewStyle, _
      , , oBaseViewOptions)
End Sub
```

Before running this sample, make sure that the file C:\TempReps.iam exists (or change the path in the sample). The file must contain a model state named MyModelState, a positional representation named MyPositionalRep and a design view representation named MyDesignViewRep.

Copy Code

```
    ' Set a reference to the drawing document.
  ' This assumes a drawing document is active.
  Dim oDrawDoc As DrawingDocument
  oDrawDoc = ThisApplication.ActiveDocument

  'Set a reference to the active sheet.
  Dim oSheet As Sheet
  oSheet = oDrawDoc.ActiveSheet

  ' Create a new NameValueMap object
  Dim oBaseViewOptions As NameValueMap
  oBaseViewOptions = ThisApplication.TransientObjects.CreateNameValueMap

  ' Set the representations to use when creating the base view.
  Call oBaseViewOptions.Add("PositionalRepresentation", "MyPositionalRep")
  Call oBaseViewOptions.Add("DesignViewRepresentation", "MyDesignViewRep")
  Call oBaseViewOptions.Add("DesignViewAssociative", True)

  ' Open the model document (corresponding to the "MyModelState" model state).
  Dim strFullDocumentName As String
  strFullDocumentName = ThisApplication.FileManager.GetFullDocumentName("C:\tempreps.iam", "MyModelState")

  Dim oModel As Document
  oModel = ThisApplication.Documents.Open(strFullDocumentName, False)

  ' Create the placement point object.
  Dim oPoint As Point2d
  oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

  ' Create a base view.
  Dim oBaseView As DrawingView
  oBaseView = oSheet.DrawingViews.AddBaseView(oModel, oPoint, 2, _
  kIsoTopLeftViewOrientation, kHiddenLineRemovedDrawingViewStyle, _
      , , oBaseViewOptions)
```

Create flat pattern drawing view

Description

This sample demonstrates the creation of a flat pattern base view in a drawing.

Code Samples

- VBA
- iLogic

Open a drawing document and run the sample. The sheet metal part must have a flat pattern within it or a folded view will still be created.

Copy Code

```
Public Sub AddFlatPatternDrawingView()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    ' Create a new NameValueMap object
    Dim oBaseViewOptions As NameValueMap
    Set oBaseViewOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Set the options to use when creating the base view.
    Call oBaseViewOptions.Add("SheetMetalFoldedModel", False)

    ' Open the sheet metal document invisibly
    Dim oModel As Document
    Set oModel = ThisApplication.Documents.Open("C:\temp\SheetMetal.ipt", False)

    ' Create the placement point object.
    Dim oPoint As Point2d
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Create a base view.
    Dim oBaseView As DrawingView
    Set oBaseView = oSheet.DrawingViews.AddBaseView(oModel, oPoint, 1, _
    kDefaultViewOrientation, kHiddenLineRemovedDrawingViewStyle, _
    , , oBaseViewOptions)
End Sub
```

Open a drawing document and run the sample. The sheet metal part must have a flat pattern within it or a folded view will still be created.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    ' Create a new NameValueMap object
    Dim oBaseViewOptions As NameValueMap
    oBaseViewOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Set the options to use when creating the base view.
    Call oBaseViewOptions.Add("SheetMetalFoldedModel", False)

    ' Open the sheet metal document invisibly
    Dim oModel As Document
    oModel = ThisApplication.Documents.Open("C:\temp\SheetMetal.ipt", False)

    ' Create the placement point object.
    Dim oPoint As Point2d
    oPoint = ThisApplication.TransientGeometry.CreatePoint2d(25, 25)

    ' Create a base view.
    Dim oBaseView As DrawingView
    oBaseView = oSheet.DrawingViews.AddBaseView(oModel, oPoint, 1, _
    kDefaultViewOrientation, kHiddenLineRemovedDrawingViewStyle, _
    , , oBaseViewOptions)
```

Create base view with representations

Description

This sample demonstrates how to create a base view by specifying various representations.

Code Samples

- C#

Before running this sample, make sure that the file C:\temp\Reps.iam exists (or change the path in the sample). The file must contain a level of detail representation named MyLODRep, a positional representation named MyPositionalRep and a design view representation named MyDesignViewRep. The first line of the C# sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void AddBaseViewWithRepresentations()
{
    Application oApp = ThisApplication;

    // Set a reference to the drawing document.
    // This assumes a drawing document is active.
    DrawingDocument oDrawDoc = (DrawingDocument)oApp.ActiveDocument;

    //Set a reference to the active sheet.
    Sheet oSheet = oDrawDoc.ActiveSheet;

    // Create a new NameValueMap object
    NameValueMap oBaseViewOptions = oApp.TransientObjects.CreateNameValueMap();

    // Set the representations to use when creating the base view.
    oBaseViewOptions.Add("PositionalRepresentation", "MyPositionalRep");
    oBaseViewOptions.Add("DesignViewRepresentation", "MyDesignViewRep");
    oBaseViewOptions.Add("DesignViewAssociative", true);

    // Open the model document (corresponding to the "MyLODRep" representation).
    String strFullDocumentName = oApp.FileManager.GetFullDocumentName("C:\temp\Reps.iam", "MyLODRep");
    Document oModel = oApp.Documents.Open(strFullDocumentName, false);

    // Create the placement point object.
    Point2d oPoint = oApp.TransientGeometry.CreatePoint2d(25, 25);

    // Create a base view.
    DrawingView oBaseView = oSheet.DrawingViews.AddBaseView((_Document)oModel, oPoint, 2,
                                       ViewOrientationTypeEnum.kIsoTopLeftViewOrientation,
                                       DrawingViewStyleEnum.kHiddenLineRemovedDrawingViewStyle,
                                       "", null, oBaseViewOptions);

    // Release reference of the invisibly open model
    oModel.ReleaseReference();

}
```

Add detail drawing view

Description

This sample demonstrates the creation of a detail drawing view with an attach point.

Code Samples

- VBA
- iLogic

Before running this sample, select a drawing view in the active drawing.

Copy Code

```
Public Sub CreateDetailView()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Select a drawing view.
    Dim oDrawingView As DrawingView
    Set oDrawingView = ThisApplication.CommandManager.Pick(kDrawingViewFilter, "Select a drawing view.")

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawingView.Parent

    ' Set a reference to the center of the base view.
    Dim oPoint As Point2d
    Set oPoint = oDrawingView.Center

    ' Translate point by a distance equal to the width of the view
    ' This will be the placement point of the detail view.
    oPoint.X = oPoint.X + oDrawingView.Width

    ' Arbitrarily find an arc within the selected drawing view.
    ' The detail view will include this arc.
    Dim oCurve As DrawingCurve
    Dim oArcCurve As DrawingCurve
    For Each oCurve In oDrawingView.DrawingCurves
        If oCurve.CurveType = kCircularArcCurve Then
            Set oArcCurve = oCurve
            Exit For
        End If
    Next

    If Not oArcCurve Is Nothing Then
        ' Use the range of the arc in sheet space to calculate the detail view box.
        Dim oCornerOne As Point2d
        Set oCornerOne = oArcCurve.Evaluator2D.RangeBox.MinPoint
        oCornerOne.X = oCornerOne.X - 1
        oCornerOne.Y = oCornerOne.Y - 1

        Dim oCornerTwo As Point2d
        Set oCornerTwo = oArcCurve.Evaluator2D.RangeBox.MaxPoint
        oCornerTwo.X = oCornerTwo.X + 1
        oCornerTwo.Y = oCornerTwo.Y + 1

        ' Create the detail view with a rectangular box.
        Dim oDetailView As DetailDrawingView
```

```
        Set oDetailView = oSheet.DrawingViews.AddDetailView(oDrawingView, oPoint, _
        kFromBaseDrawingViewStyle, False, oCornerOne, oCornerTwo, , oDrawingView.Scale * 2)
    Else
        MsgBox "No arc was found in the selected drawing view."
    End If
End Sub
```

Before running this sample, select a drawing view in the active drawing.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Select a drawing view.
    Dim oDrawingView As DrawingView
    oDrawingView = ThisApplication.CommandManager.Pick(kDrawingViewFilter, "Select a drawing view.")

    'Set a reference to the active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawingView.Parent

    ' Set a reference to the center of the base view.
    Dim oPoint As Point2d
    oPoint = oDrawingView.Center

    ' Translate point by a distance equal to the width of the view
    ' This will be the placement point of the detail view.
    oPoint.X = oPoint.X + oDrawingView.Width

    ' Arbitrarily find an arc within the selected drawing view.
    ' The detail view will include this arc.
    Dim oCurve As DrawingCurve
    Dim oArcCurve As DrawingCurve
    For Each oCurve In oDrawingView.DrawingCurves
        If oCurve.CurveType = kCircularArcCurve Then
            oArcCurve = oCurve
            Exit For
        End If
    Next

    If Not oArcCurve Is Nothing Then
        ' Use the range of the arc in sheet space to calculate the detail view box.
        Dim oCornerOne As Point2d
        oCornerOne = oArcCurve.Evaluator2D.RangeBox.MinPoint
        oCornerOne.X = oCornerOne.X - 1
        oCornerOne.Y = oCornerOne.Y - 1

        Dim oCornerTwo As Point2d
        oCornerTwo = oArcCurve.Evaluator2D.RangeBox.MaxPoint
        oCornerTwo.X = oCornerTwo.X + 1
        oCornerTwo.Y = oCornerTwo.Y + 1

        ' Create the detail view with a rectangular box.
        Dim oDetailView As DetailDrawingView
        oDetailView = oSheet.DrawingViews.AddDetailView(oDrawingView, oPoint, _
        kFromBaseDrawingViewStyle, False, oCornerOne, oCornerTwo, , oDrawingView.Scale * 2)
    Else
        MsgBox("No arc was found in the selected drawing view.")
    End If
```

Draft Views - create

Description

This sample demonstrates the creation of a draft view in a drawing.

Code Samples

- VBA
- iLogic

Open a drawing document and run the following sample.

Copy Code

```
Public Sub CreateDraftView()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    'A draft view can only be created on an active sheet.
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.ActiveSheet

    'Set a reference to the drawing view.
    Dim oDraftView As DrawingView
    Set oDraftView = oSheet.DrawingViews.AddDraftView(1#, "My Draft View")

    ' Set a reference to the sketch of the created draft view.
    Dim oSketch As DrawingSketch
    Set oSketch = oDraftView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
```

```
    Set oTG = ThisApplication.TransientGeometry

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(30, 30))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 30), oTG.CreatePoint2d(30, 10))

    ' Exit from editing the sketch.
    oSketch.ExitEdit

End Sub
```

Open a drawing document and run the following sample.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the active sheet.
    'A draft view can only be created on an active sheet.
    Dim oSheet As Sheet
    oSheet = oDrawDoc.ActiveSheet

    'Set a reference to the drawing view.
    Dim oDraftView As DrawingView
    oDraftView = oSheet.DrawingViews.AddDraftView(1#, "My Draft View")

    ' Set a reference to the sketch of the created draft view.
    Dim oSketch As DrawingSketch
    oSketch = oDraftView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Draw two lines in the sketch.
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 10), oTG.CreatePoint2d(30, 30))
    Call oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(10, 30), oTG.CreatePoint2d(30, 10))

    ' Exit from editing the sketch.
    oSketch.ExitEdit
```

Moving sketch entities to a new layer

Description

This sample demonstrates the creation of a new layer and moving sketch entities onto this newly created layer.

Code Samples

- VBA
- iLogic

To run the sample, have a drawing document open. After running the sample, select the 'Edit Layers' command in the toolbar and turn on the layer named Custom Sketch Circles.

Copy Code

```
Public Sub Layer()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the drawing view.
    Dim oDraftView As DrawingView
    Set oDraftView = oDrawDoc.ActiveSheet.DrawingViews.AddDraftView(1#, "My Draft View")

    ' Set a reference to the sketch of the created draft view.
    Dim oSketch As DrawingSketch
    Set oSketch = oDraftView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw four sketch lines that form a rectangle
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTransGeom.CreatePoint2d(10, 10), _
                                                    oTransGeom.CreatePoint2d(30, 30))

    ' Draw four sketch circles
    Dim oSketchCircles(1 To 4) As SketchCircle
    Set oSketchCircles(1) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(12, 12), 1.5)
    Set oSketchCircles(2) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(28, 12), 1.5)
    Set oSketchCircles(3) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(12, 28), 1.5)
    Set oSketchCircles(4) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(28, 28), 1.5)

    ' Create a new layer (as a copy of the 'Sketch Geometry' layer)
    ' to put the sketch circles in.
    Dim oNewLayer As Layer
    Set oNewLayer = oDrawDoc.StylesManager.Layers.Item("Sketch Geometry (ANSI)").Copy("Custom Sketch Circles")

    ' Set the LineType on the new layer to 'dashed'.
    oNewLayer.LineType = kDashedLineType

    'Put all the sketch circles on this layer
```

```
    oSketchCircles(1).Layer = oNewLayer
    oSketchCircles(2).Layer = oNewLayer
    oSketchCircles(3).Layer = oNewLayer
    oSketchCircles(4).Layer = oNewLayer

    ' Turn off the new layer
    oNewLayer.Visible = False

    ' Exit from editing the sketch.
    oSketch.ExitEdit
End Sub
```

To run the sample, have a drawing document open. After running the sample, select the 'Edit Layers' command in the toolbar and turn on the layer named Custom Sketch Circles.

Copy Code

```
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the drawing view.
    Dim oDraftView As DrawingView
    oDraftView = oDrawDoc.ActiveSheet.DrawingViews.AddDraftView(1#, "My Draft View")

    ' Set a reference to the sketch of the created draft view.
    Dim oSketch As DrawingSketch
    oSketch = oDraftView.Sketches.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw four sketch lines that form a rectangle
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTransGeom.CreatePoint2d(10, 10), _
                                                    oTransGeom.CreatePoint2d(30, 30))

    ' Draw four sketch circles
    Dim oSketchCircles(0 To 3) As SketchCircle
    oSketchCircles(0) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(12, 12), 1.5)
    oSketchCircles(1) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(28, 12), 1.5)
    oSketchCircles(2) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(12, 28), 1.5)
    oSketchCircles(3) = oSketch.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(28, 28), 1.5)

    ' Create a new layer (as a copy of the 'Sketch Geometry' layer)
    ' to put the sketch circles in.
    Dim oNewLayer As Layer
    oNewLayer = oDrawDoc.StylesManager.Layers.Item("Sketch Geometry (ANSI)").Copy("Custom Sketch Circles")

    ' Set the LineType on the new layer to 'dashed'.
    oNewLayer.LineType = kDashedLineType

    'Put all the sketch circles on this layer
    oSketchCircles(0).Layer = oNewLayer
    oSketchCircles(1).Layer = oNewLayer
    oSketchCircles(2).Layer = oNewLayer
    oSketchCircles(3).Layer = oNewLayer

    ' Turn off the new layer
    oNewLayer.Visible = False

    ' Exit from editing the sketch.
    oSketch.ExitEdit
```

Move DrawingViewAnnotation Text Position Sample

Description

This sample demonstrates how to move the text position of drawing view annotation. Create a detail or section view before running this sample.

Code Samples

- VBA
- iLogic

This sample demonstrates how to move the text position of drawing view annotation. Create a detail or section view before running this sample.

Copy Code

```
Sub MoveViewAnnotationTextPositionSample()
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oView As DrawingView
    Dim oTemp As DrawingView
    For Each oTemp In oDoc.ActiveSheet.DrawingViews
        If oTemp.ViewType = kDetailDrawingViewType Or oTemp.ViewType = kSectionDrawingViewType Then
            Set oView = oTemp

            Dim oViewAnnotation As DrawingViewAnnotation
            Set oViewAnnotation = oView.ViewAnnotation

            ' move the annotation text
            Dim oPt As Point2d
            Set oPt = oViewAnnotation.TextPosition
            oPt.X = oPt.X + 2: oPt.Y = oPt.Y + 3
            oViewAnnotation.TextPosition = oPt
```

```
            oDoc.Update
        End If
    Next
End Sub
```

This sample demonstrates how to move the text position of drawing view annotation. Create a detail or section view before running this sample.

Copy Code

```
    Dim oDoc As DrawingDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oView As DrawingView
    Dim oTemp As DrawingView
    For Each oTemp In oDoc.ActiveSheet.DrawingViews
        If oTemp.ViewType = kDetailDrawingViewType Or oTemp.ViewType = kSectionDrawingViewType Then
            oView = oTemp

            Dim oViewAnnotation As DrawingViewAnnotation
            oViewAnnotation = oView.ViewAnnotation

            ' move the annotation text
            Dim oPt As Point2d
            oPt = oViewAnnotation.TextPosition
            oPt.X = oPt.X + 2: oPt.Y = oPt.Y + 3
            oViewAnnotation.TextPosition = oPt

            oDoc.Update
        End If
    Next
```

Break alignment of a section view

Description

Sample showing how to break the alignment of a drawing section view by calling the DrawingBreakViewAlignment command.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub BreakAlignment()
   Dim oDoc As DrawingDocument
   Set oDoc = ThisApplication.ActiveDocument

   Dim oSectionView As SectionDrawingView
   Set oSectionView = oDoc.ActiveSheet.DrawingViews(2)

   oDoc.SelectSet.Clear
   oDoc.SelectSet.Select oSectionView

   Dim oCtrlDef As ControlDefinition
   Set oCtrlDef = ThisApplication.CommandManager.ControlDefinitions.Item("DrawingBreakViewAlignmentCmd")

   oCtrlDef.Execute

   oDoc.SelectSet.Clear
End Sub
```

Copy Code

```
    Dim oDoc As DrawingDocument
   oDoc = ThisApplication.ActiveDocument

   Dim oSectionView As SectionDrawingView
   oSectionView = oDoc.ActiveSheet.DrawingViews(2)

   oDoc.SelectSet.Clear
   oDoc.SelectSet.Select(oSectionView)

   Dim oCtrlDef As ControlDefinition
   oCtrlDef = ThisApplication.CommandManager.ControlDefinitions.Item("DrawingBreakViewAlignmentCmd")

   oCtrlDef.Execute

   oDoc.SelectSet.Clear
```

Create sheet with multiple views

Description

This sample demonstrates the creation of a drawing sheet based on a particular sheet format containing the definition for multiple views.

Code Samples

- VBA
- iLogic

Have a drawing document open and run the following macro.

Copy Code

```
Public Sub AddUsingSheetFormat()
    'Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the sheet format named "C Size_IPT_5 views_wNote"
    Dim oFormat As SheetFormat
    Set oFormat = oDrawDoc.SheetFormats.Item("C Size_IPT_5 views_wNote")

    'Open the model document invisible
    Dim oModel As Document
    Set oModel = ThisApplication.Documents.Open("C:\temp\block.ipt", False)

    'Create a new sheet based on the sheet format
    Dim oSheet As Sheet
    Set oSheet = oDrawDoc.Sheets.AddUsingSheetFormat(oFormat, oModel)
End Sub
```

Have a drawing document open and run the following macro.

Copy Code

```
    'Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    'Set a reference to the sheet format named "C Size_IPT_5 views_wNote"
    Dim oFormat As SheetFormat
    oFormat = oDrawDoc.SheetFormats.Item("C Size_IPT_5 views_wNote")

    'Open the model document invisible
    Dim oModel As Document
    oModel = ThisApplication.Documents.Open("C:\temp\block.ipt", False)

    'Create a new sheet based on the sheet format
    Dim oSheet As Sheet
    oSheet = oDrawDoc.Sheets.AddUsingSheetFormat(oFormat, oModel)
```

Client Graphics - Draw Range Box

Description

This sample demonstrates the use of client graphics to draw the range box of selected entities.

Code Samples

- VBA
- iLogic

To run this sample open an assembly or part document and select the item(s) to draw a range box around. Then run the program. If you run it without any graphics selected, it will delete any existing range boxes.

Copy Code

```
Public Sub DrawRangeBox()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Make sure something is selected.
    If oDoc.SelectSet.Count = 0 Then
        MsgBox "You must select the object(s) to display the range box."

        ' Delete any graphics, if they exist.
        On Error Resume Next
        Dim oExistingGraphicsData As GraphicsDataSets
        Set oExistingGraphicsData = oDoc.GraphicsDataSetsCollection.Item("RangeBoxGraphics")
        If Err.Number = 0 Then
            On Error Goto 0
            Dim oExistingGraphics As ClientGraphics
            Set oExistingGraphics = oCompDef.ClientGraphicsCollection.Item("RangeBoxGraphics")
            oExistingGraphics.Delete
            oExistingGraphicsData.Delete
            ThisApplication.ActiveView.Update
        End If

        Exit Sub
    End If

    Redim aoRanges(1 To oDoc.SelectSet.Count) As Box
    Dim iRangeCount As Long
    Dim i As Long
    On Error Resume Next
    For i = 1 To oDoc.SelectSet.Count
       Dim oBox As Box
```

```
    Set oBox = oDoc.SelectSet.Item(i).RangeBox
    If Err Then
        Err.Clear
        ' Special case for B-Rep entities.
        If oDoc.SelectSet.Item(i).Type = kFaceObject Or _
           oDoc.SelectSet.Item(i).Type = kFaceProxyObject Or _
           oDoc.SelectSet.Item(i).Type = kEdgeObject Or _
           oDoc.SelectSet.Item(i).Type = kEdgeProxyObject Then
            ' Get the range from evaluator of the BRep object.
            Set oBox = oDoc.SelectSet.Item(i).Evaluator.RangeBox
            iRangeCount = iRangeCount + 1
            Set aoRanges(iRangeCount) = oBox
        End If
    Else
        iRangeCount = iRangeCount + 1
        Set aoRanges(iRangeCount) = oBox
    End If
Next
On Error Goto 0

If iRangeCount = 0 Then
    MsgBox "You must pick object(s) that support a 3D RangeBox property."
    Exit Sub
End If

' Check to see if range box graphics information already exists.
On Error Resume Next
Dim oClientGraphics As ClientGraphics
Dim oLineGraphics As LineGraphics
Dim oBoxNode As GraphicsNode
Dim oGraphicsData As GraphicsDataSets
Set oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("RangeBoxGraphics")
If Err Then
    Err.Clear
    On Error Goto 0

    ' Set a reference to the transient geometry object for user later.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a graphics data set object. This object contains all of the
    ' information used to define the graphics.
    Dim oDataSets As GraphicsDataSets
    Set oDataSets = oDoc.GraphicsDataSetsCollection.Add("RangeBoxGraphics")

    ' Create a coordinate set.
    Dim oCoordSet As GraphicsCoordinateSet
    Set oCoordSet = oDataSets.CreateCoordinateSet(1)

    ' Create the client graphics for this compdef.
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("RangeBoxGraphics")

    ' Create a graphics node.
    Set oBoxNode = oClientGraphics.AddNode(1)
    oBoxNode.Selectable = False

    ' Create line graphics.
    Set oLineGraphics = oBoxNode.AddLineGraphics
    oLineGraphics.CoordinateSet = oCoordSet
Else
    Set oCoordSet = oGraphicsData.ItemById(1)
    Set oBoxNode = oCompDef.ClientGraphicsCollection.Item("RangeBoxGraphics").ItemById(1)
End If

Dim dBoxLines() As Double
Redim dBoxLines(1 To 12 * 6 * iRangeCount) As Double
For i = 0 To iRangeCount - 1
    Dim MinPoint(1 To 3) As Double
    Dim MaxPoint(1 To 3) As Double
    Call aoRanges(i + 1).GetBoxData(MinPoint, MaxPoint)

    ' Line 1
    dBoxLines(i * 72 + 1) = MinPoint(1)
    dBoxLines(i * 72 + 2) = MinPoint(2)
    dBoxLines(i * 72 + 3) = MinPoint(3)
    dBoxLines(i * 72 + 4) = MaxPoint(1)
    dBoxLines(i * 72 + 5) = MinPoint(2)
    dBoxLines(i * 72 + 6) = MinPoint(3)

    ' Line 2
    dBoxLines(i * 72 + 7) = MinPoint(1)
    dBoxLines(i * 72 + 8) = MinPoint(2)
    dBoxLines(i * 72 + 9) = MinPoint(3)
    dBoxLines(i * 72 + 10) = MinPoint(1)
    dBoxLines(i * 72 + 11) = MaxPoint(2)
    dBoxLines(i * 72 + 12) = MinPoint(3)

    ' Line 3
    dBoxLines(i * 72 + 13) = MinPoint(1)
    dBoxLines(i * 72 + 14) = MinPoint(2)
    dBoxLines(i * 72 + 15) = MinPoint(3)
    dBoxLines(i * 72 + 16) = MinPoint(1)
    dBoxLines(i * 72 + 17) = MinPoint(2)
    dBoxLines(i * 72 + 18) = MaxPoint(3)

    ' Line 4
    dBoxLines(i * 72 + 19) = MaxPoint(1)
    dBoxLines(i * 72 + 20) = MaxPoint(2)
    dBoxLines(i * 72 + 21) = MaxPoint(3)
    dBoxLines(i * 72 + 22) = MinPoint(1)
    dBoxLines(i * 72 + 23) = MaxPoint(2)
    dBoxLines(i * 72 + 24) = MaxPoint(3)

    ' Line 5
    dBoxLines(i * 72 + 25) = MaxPoint(1)
    dBoxLines(i * 72 + 26) = MaxPoint(2)
    dBoxLines(i * 72 + 27) = MaxPoint(3)
```

```
        dBoxLines(i * 72 + 28) = MaxPoint(1)
        dBoxLines(i * 72 + 29) = MinPoint(2)
        dBoxLines(i * 72 + 30) = MaxPoint(3)

        ' Line 6
        dBoxLines(i * 72 + 31) = MaxPoint(1)
        dBoxLines(i * 72 + 32) = MaxPoint(2)
        dBoxLines(i * 72 + 33) = MaxPoint(3)
        dBoxLines(i * 72 + 34) = MaxPoint(1)
        dBoxLines(i * 72 + 35) = MaxPoint(2)
        dBoxLines(i * 72 + 36) = MinPoint(3)

        ' Line 7
        dBoxLines(i * 72 + 37) = MinPoint(1)
        dBoxLines(i * 72 + 38) = MaxPoint(2)
        dBoxLines(i * 72 + 39) = MinPoint(3)
        dBoxLines(i * 72 + 40) = MaxPoint(1)
        dBoxLines(i * 72 + 41) = MaxPoint(2)
        dBoxLines(i * 72 + 42) = MinPoint(3)

        ' Line 8
        dBoxLines(i * 72 + 43) = MinPoint(1)
        dBoxLines(i * 72 + 44) = MaxPoint(2)
        dBoxLines(i * 72 + 45) = MinPoint(3)
        dBoxLines(i * 72 + 46) = MinPoint(1)
        dBoxLines(i * 72 + 47) = MaxPoint(2)
        dBoxLines(i * 72 + 48) = MaxPoint(3)

        ' Line 9
        dBoxLines(i * 72 + 49) = MaxPoint(1)
        dBoxLines(i * 72 + 50) = MinPoint(2)
        dBoxLines(i * 72 + 51) = MaxPoint(3)
        dBoxLines(i * 72 + 52) = MaxPoint(1)
        dBoxLines(i * 72 + 53) = MinPoint(2)
        dBoxLines(i * 72 + 54) = MinPoint(3)

        ' Line 10
        dBoxLines(i * 72 + 55) = MaxPoint(1)
        dBoxLines(i * 72 + 56) = MinPoint(2)
        dBoxLines(i * 72 + 57) = MaxPoint(3)
        dBoxLines(i * 72 + 58) = MinPoint(1)
        dBoxLines(i * 72 + 59) = MinPoint(2)
        dBoxLines(i * 72 + 60) = MaxPoint(3)

        ' Line 11
        dBoxLines(i * 72 + 61) = MinPoint(1)
        dBoxLines(i * 72 + 62) = MinPoint(2)
        dBoxLines(i * 72 + 63) = MaxPoint(3)
        dBoxLines(i * 72 + 64) = MinPoint(1)
        dBoxLines(i * 72 + 65) = MaxPoint(2)
        dBoxLines(i * 72 + 66) = MaxPoint(3)

        ' Line 12
        dBoxLines(i * 72 + 67) = MaxPoint(1)
        dBoxLines(i * 72 + 68) = MinPoint(2)
        dBoxLines(i * 72 + 69) = MinPoint(3)
        dBoxLines(i * 72 + 70) = MaxPoint(1)
        dBoxLines(i * 72 + 71) = MaxPoint(2)
        dBoxLines(i * 72 + 72) = MinPoint(3)
    Next

    ' Assign the points into the coordinate set.
    Call oCoordSet.PutCoordinates(dBoxLines)

    ' Update the display.
    ThisApplication.ActiveView.Update
End Sub
```

To run this sample open an assembly or part document and select the item(s) to draw a range box around. Then run the program. If you run it without any graphics selected, it will delete any existing range boxes.

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Make sure something is selected.
    If oDoc.SelectSet.Count = 0 Then
        MsgBox("You must select the object(s) to display the range box.")

        ' Delete any graphics, if they exist.
        On Error Resume Next
        Dim oExistingGraphicsData As GraphicsDataSets
        oExistingGraphicsData = oDoc.GraphicsDataSetsCollection.Item("RangeBoxGraphics")
        If Err.Number = 0 Then
            On Error Goto 0
            Dim oExistingGraphics As ClientGraphics
            oExistingGraphics = oCompDef.ClientGraphicsCollection.Item("RangeBoxGraphics")
            oExistingGraphics.Delete
            oExistingGraphicsData.Delete
            ThisApplication.ActiveView.Update
        End If

        Exit Sub
    End If

    Dim aoRanges(0 To oDoc.SelectSet.Count-1) As Box
    Dim iRangeCount As Long
    Dim i As Long
    On Error Resume Next
    For i = 1 To oDoc.SelectSet.Count
       Dim oBox As Box
```

```
    oBox = oDoc.SelectSet.Item(i).RangeBox
    If Err.Number > 0 Then
        Err.Clear
        ' Special case for B-Rep entities.
        If oDoc.SelectSet.Item(i).Type = kFaceObject Or _
           oDoc.SelectSet.Item(i).Type = kFaceProxyObject Or _
           oDoc.SelectSet.Item(i).Type = kEdgeObject Or _
           oDoc.SelectSet.Item(i).Type = kEdgeProxyObject Then
            ' Get the range from evaluator of the BRep object.
            oBox = oDoc.SelectSet.Item(i).Evaluator.RangeBox
            iRangeCount = iRangeCount + 1
            aoRanges(iRangeCount-1) = oBox
        End If
    Else
        iRangeCount = iRangeCount + 1
        aoRanges(iRangeCount-1) = oBox
    End If
Next
On Error Goto 0

If iRangeCount = 0 Then
    MsgBox("You must pick object(s) that support a 3D RangeBox property.")
    Exit Sub
End If

' Check to see if range box graphics information already exists.
On Error Resume Next
Dim oClientGraphics As ClientGraphics
Dim oLineGraphics As LineGraphics
Dim oBoxNode As GraphicsNode
Dim oGraphicsData As GraphicsDataSets
oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("RangeBoxGraphics")

Dim oCoordSet As GraphicsCoordinateSet
If Err.Number > 0 Then
    Err.Clear
    On Error Goto 0

    ' Set a reference to the transient geometry object for user later.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a graphics data set object. This object contains all of the
    ' information used to define the graphics.
    Dim oDataSets As GraphicsDataSets
    oDataSets = oDoc.GraphicsDataSetsCollection.Add("RangeBoxGraphics")

    ' Create a coordinate set.
    oCoordSet = oDataSets.CreateCoordinateSet(1)

    ' Create the client graphics for this compdef.
    oClientGraphics = oCompDef.ClientGraphicsCollection.Add("RangeBoxGraphics")

    ' Create a graphics node.
    oBoxNode = oClientGraphics.AddNode(1)
    oBoxNode.Selectable = False

    ' Create line graphics.
    oLineGraphics = oBoxNode.AddLineGraphics
    oLineGraphics.CoordinateSet = oCoordSet
Else
    oCoordSet = oGraphicsData.ItemById(1)
    oBoxNode = oCompDef.ClientGraphicsCollection.Item("RangeBoxGraphics").ItemById(1)
End If

Dim  dBoxLines(0 To 12 * 6 * iRangeCount-1) As Double
For i = 0 To iRangeCount - 1
    Dim MinPoint(0 To 2) As Double
    Dim MaxPoint(0 To 2) As Double
    Call aoRanges(i).GetBoxData(MinPoint, MaxPoint)

    ' Line 1
    dBoxLines(i * 72 + 0) = MinPoint(0)
    dBoxLines(i * 72 + 1) = MinPoint(1)
    dBoxLines(i * 72 + 2) = MinPoint(2)
    dBoxLines(i * 72 + 3) = MaxPoint(0)
    dBoxLines(i * 72 + 4) = MinPoint(1)
    dBoxLines(i * 72 + 5) = MinPoint(2)

    ' Line 2
    dBoxLines(i * 72 + 6) = MinPoint(0)
    dBoxLines(i * 72 + 7) = MinPoint(1)
    dBoxLines(i * 72 + 8) = MinPoint(2)
    dBoxLines(i * 72 + 9) = MinPoint(0)
    dBoxLines(i * 72 + 10) = MaxPoint(1)
    dBoxLines(i * 72 + 11) = MinPoint(2)

    ' Line 3
    dBoxLines(i * 72 + 12) = MinPoint(0)
    dBoxLines(i * 72 + 13) = MinPoint(1)
    dBoxLines(i * 72 + 14) = MinPoint(2)
    dBoxLines(i * 72 + 15) = MinPoint(0)
    dBoxLines(i * 72 + 16) = MinPoint(1)
    dBoxLines(i * 72 + 17) = MaxPoint(2)

    ' Line 4
    dBoxLines(i * 72 + 18) = MaxPoint(0)
    dBoxLines(i * 72 + 19) = MaxPoint(1)
    dBoxLines(i * 72 + 20) = MaxPoint(2)
    dBoxLines(i * 72 + 21) = MinPoint(0)
    dBoxLines(i * 72 + 22) = MaxPoint(1)
    dBoxLines(i * 72 + 23) = MaxPoint(2)

    ' Line 5
    dBoxLines(i * 72 + 24) = MaxPoint(0)
    dBoxLines(i * 72 + 25) = MaxPoint(1)
    dBoxLines(i * 72 + 26) = MaxPoint(2)
```

```
        dBoxLines(i * 72 + 27) = MaxPoint(0)
        dBoxLines(i * 72 + 28) = MinPoint(1)
        dBoxLines(i * 72 + 29) = MaxPoint(2)

        ' Line 6
        dBoxLines(i * 72 + 30) = MaxPoint(0)
        dBoxLines(i * 72 + 31) = MaxPoint(1)
        dBoxLines(i * 72 + 32) = MaxPoint(2)
        dBoxLines(i * 72 + 33) = MaxPoint(0)
        dBoxLines(i * 72 + 34) = MaxPoint(1)
        dBoxLines(i * 72 + 35) = MinPoint(2)

        ' Line 7
        dBoxLines(i * 72 + 36) = MinPoint(0)
        dBoxLines(i * 72 + 37) = MaxPoint(1)
        dBoxLines(i * 72 + 38) = MinPoint(2)
        dBoxLines(i * 72 + 39) = MaxPoint(0)
        dBoxLines(i * 72 + 40) = MaxPoint(1)
        dBoxLines(i * 72 + 41) = MinPoint(2)

        ' Line 8
        dBoxLines(i * 72 + 42) = MinPoint(0)
        dBoxLines(i * 72 + 43) = MaxPoint(1)
        dBoxLines(i * 72 + 44) = MinPoint(2)
        dBoxLines(i * 72 + 45) = MinPoint(0)
        dBoxLines(i * 72 + 46) = MaxPoint(1)
        dBoxLines(i * 72 + 47) = MaxPoint(2)

        ' Line 9
        dBoxLines(i * 72 + 48) = MaxPoint(0)
        dBoxLines(i * 72 + 49) = MinPoint(1)
        dBoxLines(i * 72 + 50) = MaxPoint(2)
        dBoxLines(i * 72 + 51) = MaxPoint(0)
        dBoxLines(i * 72 + 52) = MinPoint(1)
        dBoxLines(i * 72 + 53) = MinPoint(2)

        ' Line 10
        dBoxLines(i * 72 + 54) = MaxPoint(0)
        dBoxLines(i * 72 + 55) = MinPoint(1)
        dBoxLines(i * 72 + 56) = MaxPoint(2)
        dBoxLines(i * 72 + 57) = MinPoint(0)
        dBoxLines(i * 72 + 58) = MinPoint(1)
        dBoxLines(i * 72 + 59) = MaxPoint(2)

        ' Line 11
        dBoxLines(i * 72 + 60) = MinPoint(0)
        dBoxLines(i * 72 + 61) = MinPoint(1)
        dBoxLines(i * 72 + 62) = MaxPoint(2)
        dBoxLines(i * 72 + 63) = MinPoint(0)
        dBoxLines(i * 72 + 64) = MaxPoint(1)
        dBoxLines(i * 72 + 65) = MaxPoint(2)

        ' Line 12
        dBoxLines(i * 72 + 66) = MaxPoint(0)
        dBoxLines(i * 72 + 67) = MinPoint(1)
        dBoxLines(i * 72 + 68) = MinPoint(2)
        dBoxLines(i * 72 + 69) = MaxPoint(0)
        dBoxLines(i * 72 + 70) = MaxPoint(1)
        dBoxLines(i * 72 + 71) = MinPoint(2)
    Next

    ' Assign the points into the coordinate set.
    Call oCoordSet.PutCoordinates(dBoxLines)

    ' Update the display.
    ThisApplication.ActiveView.Update
```

Client graphics texture-based color mapping

Description

This test applies texture coordinates expressing distance from the origin to 'the triangle mesh of whatever Part you have open. It then creates either a discrete-band or continuous color mapper and allows you to adjust the values of the mapper to change the range of values that map to various colors.

Code Samples

- VBA
- iLogic

To operate: 1. Open a part 2. Run Demo - it should show discrete bands moving across the model 3. Click on any of the other "convenience functions" and hit F5 to exercise them.

Copy Code

```
Dim lastOffset As Double
Dim mapperType As Integer

'convenience "links"
Sub Demo()
    ThisApplication.ActiveView.DisplayMode = kWireframeRendering
    Call OffsetSurfaceBy(0#)
    Call UpdateValues(1 / 1.01, 0, 30)
    Call UpdateValues(1, -0.01, 20)
    Call UpdateValues(1, 0.01, 20)
    Call UpdateValues(1 / 1.01, 0, 50)
    Call UpdateValues(1, -0.01, 20)
    Call UpdateValues(1, 0.01, 20)
    Call UpdateValues(1 * 1.01, 0, 50)
```

```
End Sub

Sub narrowBands()
    Call UpdateValues(1 / 1.01, 0, 1)
End Sub

Sub widenBands()
    Call UpdateValues(1 * 1.01, 0, 1)
End Sub

Sub slideBandsUp()
    Call UpdateValues(1, 0.01, 1)
End Sub

Sub slideBandsDown()
    Call UpdateValues(1, -0.01, 1)
End Sub

Sub setContinuousMode()
    mapperType = 1
    Call OffsetSurfaceBy(0#)
End Sub

Sub setDiscreteMode()
    mapperType = 0
    Call OffsetSurfaceBy(0#)
End Sub

'internal functions
'randomly change colors in ColorMapper
Sub UpdateColors()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDataSets As GraphicsDataSets
    Set oDataSets = oDoc.GraphicsDataSetsCollection.Item("CG_Test")

    Dim oCompDef As ComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("CG_Test")

    Dim oGraphicsNode As GraphicsNode
    Set oGraphicsNode = oClientGraphics.ItemById(100)

    Dim oTriangleSet As TriangleGraphics
    Set oTriangleSet = oGraphicsNode.Item(1)
    Dim oColorMapper As GraphicsColorMapper
    Set oColorMapper = oTriangleSet.ColorMapper

    For i = 1 To 100
        Dim cmColors() As Byte
        Call oColorMapper.GetColors(cmColors)
        For j = 0 To oColorMapper.ColorCount * 3 - 1
            cmColors(j) = 255 * Rnd() * i / 100#
        Next
        Call oColorMapper.PutColors(cmColors)
        ThisApplication.ActiveView.Update
    Next
End Sub

'change values in ColorMapper
Sub UpdateValues(factor As Double, offset As Double, count As Integer)
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDataSets As GraphicsDataSets
    Set oDataSets = oDoc.GraphicsDataSetsCollection.Item("CG_Test")

    Dim oCompDef As ComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("CG_Test")

    Dim oGraphicsNode As GraphicsNode
    Set oGraphicsNode = oClientGraphics.ItemById(100)

    Dim oTriangleSet As TriangleGraphics
    Set oTriangleSet = oGraphicsNode.Item(1)
    Dim oColorMapper As GraphicsColorMapper
    Set oColorMapper = oTriangleSet.ColorMapper

    For i = 1 To count
        Dim v() As Double
        Call oColorMapper.GetValues(v)
        Dim vMid As Double
        vMid = v(oColorMapper.ValueCount / 2)
        For j = 0 To oColorMapper.ValueCount - 1
            v(j) = (v(j) - vMid) * factor + vMid + offset * (v(oColorMapper.ValueCount - 1) - v(0))
        Next
        Call oColorMapper.PutValues(v)
        ThisApplication.ActiveView.Update
    Next
End Sub

Function min(a, b)
    min = a
    If (b < a) Then
        min = b
    End If
End Function

Function max(a, b)
    max = a
    If (b > a) Then
```

```
        max = b
    End If
End Function

'display part thickness by adding display of surfaces offset by given distance --
'wherever they stick through the part, it's thinner than that distance at that point
Public Sub OffsetSurfaceBy(offset As Double)
    ' Get the surface body from the active document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    Dim oSurfBody As SurfaceBody
    Set oSurfBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)
    Set oSurfBody = oPartDoc.ComponentDefinitions.Item(1).SurfaceBodies.Item(1)

    ' Delete the graphics data set and client graphics, if they exist.
    Dim oDataSets As GraphicsDataSets
    On Error Resume Next
    Set oDataSets = oPartDoc.GraphicsDataSetsCollection.Item("CG_Test")
    If Err.Number = 0 Then
        oDataSets.Delete
        oPartDoc.ComponentDefinition.ClientGraphicsCollection.Item("CG_Test").Delete
        oSurfBody.Visible = True
        ThisApplication.ActiveView.Update
    End If
    On Error GoTo 0

'    If offset = 0 Then
'        Exit Sub
'    End If

    ' Determine the highest tolerance of the existing facet sets.
    Dim ToleranceCount As Long
    Dim ExistingTolerances() As Double
    Call oSurfBody.GetExistingFacetTolerances(ToleranceCount, ExistingTolerances)

    Dim i As Long
    Dim BestTolerance As Double
    For i = 0 To ToleranceCount - 1
        If i = 0 Then
            BestTolerance = ExistingTolerances(i)
        ElseIf ExistingTolerances(i) < BestTolerance Then
            BestTolerance = ExistingTolerances(i)
        End If
    Next

    ' Get a set of existing facets.
    Dim iVertexCount As Long
    Dim iFacetCount As Long
    Dim adVertexCoords() As Double
    Dim adNormalVectors() As Double
    Dim aiVertexIndices() As Long
    Call oSurfBody.GetExistingFacets(BestTolerance, iVertexCount, iFacetCount, _
                                     adVertexCoords, adNormalVectors, aiVertexIndices)

    ' Offset vertices by given distance along anti-normals
    For i = 0 To (iVertexCount * 3 - 1)
        adVertexCoords(i) = adVertexCoords(i) - adNormalVectors(i) * offset
    Next

    ' Start a transaction.
    Dim oTrans As Transaction
    Set oTrans = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Z Height Colors")

    ' Create the graphics data sets collection.
    Set oDataSets = oPartDoc.GraphicsDataSetsCollection.Add("CG_Test")

    ' Create the coordinate set and set it using the coordinates from the facets.
    Dim oGraphicsCoordSet As GraphicsCoordinateSet
    Set oGraphicsCoordSet = oDataSets.CreateCoordinateSet(1)
    Call oGraphicsCoordSet.PutCoordinates(adVertexCoords)

    ' Create the index set and set it using the indices from the facets.
    Dim oGraphicsIndexSet As GraphicsIndexSet
    Set oGraphicsIndexSet = oDataSets.CreateIndexSet(2)
    Call oGraphicsIndexSet.PutIndices(aiVertexIndices)

    ' Create the normal set and set it using the normals from the facets.
    Dim oGraphicsNormalSet As GraphicsNormalSet
    Set oGraphicsNormalSet = oDataSets.CreateNormalSet(3)
    Call oGraphicsNormalSet.PutNormals(adNormalVectors)

    ' Allocate the array that will contain the color information.
    ' This array contains RGB values for each vertex.
    Dim abtColors() As Byte
    ReDim abtColors(0 To iVertexCount * 3 - 1) As Byte

    ' Load the array with color information for each vertex.
    For i = 0 To iVertexCount - 1
        ' Set the color information for the current vertex.
        ' currently, all vertices are a constant color, but ...
        abtColors(i * 3) = 200
        abtColors(i * 3 + 1) = 0
        abtColors(i * 3 + 2) = 0
    Next

    ' Create the color set and set it using the array of rgb values just created.
    Dim oGraphicsColorSet As GraphicsColorSet
    Set oGraphicsColorSet = oDataSets.CreateColorSet(4)
    Call oGraphicsColorSet.PutColors(abtColors)

    ' Create a scalar data texture coordinate set representing distance from origin
    Dim oTCSet As GraphicsTextureCoordinateSet
    Set oTCSet = oDataSets.CreateTextureCoordinateSet(5)
    Dim tc() As Double
    ReDim tc(1 To iVertexCount) As Double
    Dim tcMax As Double
    tcMax = 0
```

```
    Dim tcMin As Double
    tcMin = 1000000#
    For i = 1 To iVertexCount
        tc(i) = Sqr(adVertexCoords(3 * (i - 1)) * adVertexCoords(3 * (i - 1)) + adVertexCoords(3 * (i - 1) + 1) * adVertexCoords(3 * (
        tcMin = min(tcMin, tc(i))
        tcMax = max(tcMax, tc(i))
    Next
    Call oTCSet.PutCoordinates(tc)

    ' Create the client graphics collection.
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oPartDoc.ComponentDefinition.ClientGraphicsCollection.Add("CG_Test")

    ' Create a graphics node.
    Dim oGraphicNode As GraphicsNode
    Set oGraphicNode = oClientGraphics.AddNode(100)

    ' Create the triangle graphics.
    Dim oTriangles As TriangleGraphics
    Set oTriangles = oGraphicNode.AddTriangleGraphics

    '============================================

    Dim oColorMapper As GraphicsColorMapper
    Set oColorMapper = oDataSets.CreateColorMapper

    '============================================

    'construct ColorMapper:
    '  black - red - orange - yellow - green - cyan - blue - magenta - black
    '      minV                          ...                         maxV
    Dim nColors As Integer
    nColors = 9

    Dim cmColors() As Byte
    ReDim cmColors(1 To nColors * 3) As Byte
    For i = 1 To nColors * 3
        cmColors(i) = 0
    Next
    'black
    cmColors(4) = 255       'red
    cmColors(7) = 255       'orange
    cmColors(7 + 1) = 125
    cmColors(10) = 255      'yellow
    cmColors(10 + 1) = 255
    cmColors(13 + 1) = 255  'green
    cmColors(16 + 1) = 255  'cyan
    cmColors(16 + 2) = 255
    cmColors(19 + 2) = 255  'blue
    cmColors(22) = 255      'magenta
    cmColors(22 + 2) = 255
    'black

    Dim nValues As Integer
    If (mapperType = 0) Then
        nValues = nColors - 1   'discrete value boundaries
    Else
        nValues = nColors       'continuous value points
    End If

    Dim cmValues() As Double
    ReDim cmValues(1 To nValues) As Double
    For i = 1 To nValues
        cmValues(i) = tcMin + (tcMax - tcMin) * (i - 1#) / (nValues - 1#)
    Next

    '============================================

    Call oColorMapper.PutColors(cmColors)
    Call oColorMapper.PutValues(cmValues)


    ' Set various properties of the triangle graphics.
    oTriangles.CoordinateSet = oGraphicsCoordSet
    oTriangles.CoordinateIndexSet = oGraphicsIndexSet
    oTriangles.NormalSet = oGraphicsNormalSet
    oTriangles.NormalBinding = kPerVertexNormals
    oTriangles.NormalIndexSet = oGraphicsIndexSet
'    oTriangles.ColorSet = oGraphicsColorSet
'    oTriangles.ColorBinding = kPerVertexColors
'    oTriangles.ColorIndexSet = oGraphicsIndexSet
    oTriangles.TextureCoordinateSet = oTCSet
    oTriangles.TextureCoordinateIndexSet = oGraphicsIndexSet
    oTriangles.ColorMapper = oColorMapper

    ' End the transaction.
    oTrans.End

    ' Update the view.
    ThisApplication.ActiveView.Update
End Sub

Public Sub increaseOffset()
    lastOffset = lastOffset * 1.1
    Dim offset As Double
    offset = lastOffset
    Call OffsetSurfaceBy(offset)
End Sub

Public Sub DecreaseOffset()
    lastOffset = lastOffset * 0.9
    Dim offset As Double
    offset = lastOffset
    Call OffsetSurfaceBy(offset)
End Sub

Public Sub OffsetSurface()
```

```
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    Dim lenunits As UnitsTypeEnum
    lenunits = oPartDoc.UnitsOfMeasure.LengthUnits
    Dim unitscale As Double
    Dim unitname As String
    If lenunits = kInchLengthUnits Then
        unitscale = 2.54
        unitname = "inches"
    Else
    If lenunits = kMillimeterLengthUnits Then
        unitscale = 0.1
        unitname = "millimeters"
    Else
    If lenunits = kCentimeterLengthUnits Then
        unitscale = 1#
        unitname = "centimeters"
    End If
    End If
    End If

    Dim offset As Double
    offset = 0
    On Error Resume Next
    offset = InputBox("Enter offset (in " + unitname + "):", "Offset", lastOffset / unitscale)
    On Error GoTo 0

    lastOffset = offset * unitscale

    Call OffsetSurfaceBy(lastOffset)
End Sub
```

To operate: 1. Open a part 2. Run the sample - it should show discrete bands moving across the model.

Copy Code

```
Class Test
        Dim lastOffset As Double
        Dim mapperType As Integer

        'convenience "links"
        Sub Main
            ThisApplication.ActiveView.DisplayMode = kWireframeRendering
            Call OffsetSurfaceBy(0#)
            Call UpdateValues(1 / 1.01, 0, 30)
            Call UpdateValues(1, -0.01, 20)
            Call UpdateValues(1, 0.01, 20)
            Call UpdateValues(1 / 1.01, 0, 50)
            Call UpdateValues(1, -0.01, 20)
            Call UpdateValues(1, 0.01, 20)
            Call UpdateValues(1 * 1.01, 0, 50)

                narrowBands()
                widenBands()
                slideBandsUp()
                slideBandsDown()
                setContinuousMode()
                setDiscreteMode()

        End Sub

        Sub narrowBands()
            Call UpdateValues(1 / 1.01, 0, 1)
        End Sub

        Sub widenBands()
            Call UpdateValues(1 * 1.01, 0, 1)
        End Sub

        Sub slideBandsUp()
            Call UpdateValues(1, 0.01, 1)
        End Sub

        Sub slideBandsDown()
            Call UpdateValues(1, -0.01, 1)
        End Sub

        Sub setContinuousMode()
            mapperType = 1
            Call OffsetSurfaceBy(0#)
        End Sub

        Sub setDiscreteMode()
            mapperType = 0
            Call OffsetSurfaceBy(0#)
        End Sub

        'internal functions
        'randomly change colors in ColorMapper
        Sub UpdateColors()
            Dim oDoc As Document
            oDoc = ThisApplication.ActiveDocument

            Dim oDataSets As GraphicsDataSets
            oDataSets = oDoc.GraphicsDataSetsCollection.Item("CG_Test")

            Dim oCompDef As ComponentDefinition
            oCompDef = oDoc.ComponentDefinition

            Dim oClientGraphics As ClientGraphics
            oClientGraphics = oCompDef.ClientGraphicsCollection.Item("CG_Test")

            Dim oGraphicsNode As GraphicsNode
            oGraphicsNode = oClientGraphics.ItemById(100)

            Dim oTriangleSet As TriangleGraphics
            oTriangleSet = oGraphicsNode.Item(1)
```

```
    Dim oColorMapper As GraphicsColorMapper
    oColorMapper = oTriangleSet.ColorMapper

    For i = 1 To 100
        Dim cmColors() As Byte
        Call oColorMapper.GetColors(cmColors)
        For j = 0 To oColorMapper.ColorCount * 3 - 1
            cmColors(j) = 255 * Rnd() * i / 100#
        Next
        Call oColorMapper.PutColors(cmColors)
        ThisApplication.ActiveView.Update
    Next
End Sub

'change values in ColorMapper
Sub UpdateValues(factor As Double, offset As Double, count As Integer)
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    Dim oDataSets As GraphicsDataSets
    oDataSets = oDoc.GraphicsDataSetsCollection.Item("CG_Test")

    Dim oCompDef As ComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oCompDef.ClientGraphicsCollection.Item("CG_Test")

    Dim oGraphicsNode As GraphicsNode
    oGraphicsNode = oClientGraphics.ItemById(100)

    Dim oTriangleSet As TriangleGraphics
    oTriangleSet = oGraphicsNode.Item(1)
    Dim oColorMapper As GraphicsColorMapper
    oColorMapper = oTriangleSet.ColorMapper

    For i = 1 To count
        Dim v() As Double = New Double() {}
        Call oColorMapper.GetValues(v)
        Dim vMid As Double
        vMid = v(oColorMapper.ValueCount / 2)
        For j = 0 To oColorMapper.ValueCount - 1
            v(j) = (v(j) - vMid) * factor + vMid + offset * (v(oColorMapper.ValueCount - 1) - v(0))
        Next
        Call oColorMapper.PutValues(v)
        ThisApplication.ActiveView.Update
    Next
End Sub

Function min(a, b)
    min = a
    If (b < a) Then
        min = b
    End If
End Function

Function max(a, b)
    max = a
    If (b > a) Then
        max = b
    End If
End Function

'display part thickness by adding display of surfaces offset by given distance --
'wherever they stick through the part, it's thinner than that distance at that point
Public Sub OffsetSurfaceBy(offset As Double)
    ' Get the surface body from the active document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    Dim oSurfBody As SurfaceBody
    oSurfBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)
    oSurfBody = oPartDoc.ComponentDefinitions.Item(1).SurfaceBodies.Item(1)

    ' Delete the graphics data set and client graphics, if they exist.
    Dim oDataSets As GraphicsDataSets
    On Error Resume Next
    oDataSets = oPartDoc.GraphicsDataSetsCollection.Item("CG_Test")
    If Err.Number = 0 Then
        oDataSets.Delete
        oPartDoc.ComponentDefinition.ClientGraphicsCollection.Item("CG_Test").Delete
        oSurfBody.Visible = True
        ThisApplication.ActiveView.Update
    End If
    On Error GoTo 0

'    If offset = 0 Then
'        Exit Sub
'    End If

    ' Determine the highest tolerance of the existing facet sets.
    Dim ToleranceCount As Long
    Dim ExistingTolerances() As Double = New Double() {}
    Call oSurfBody.GetExistingFacetTolerances(ToleranceCount, ExistingTolerances)

    Dim i As Long
    Dim BestTolerance As Double
    For i = 0 To ToleranceCount - 1
        If i = 0 Then
            BestTolerance = ExistingTolerances(i)
        ElseIf ExistingTolerances(i) < BestTolerance Then
            BestTolerance = ExistingTolerances(i)
        End If
    Next

    ' Get a set of existing facets.
    Dim iVertexCount As Integer
    Dim iFacetCount As Integer
```

```
Dim adVertexCoords() As Double = New Double() {}
Dim adNormalVectors() As Double = New Double() {}
Dim aiVertexIndices() As Integer = New Integer() {}
Call oSurfBody.GetExistingFacets(BestTolerance, iVertexCount, iFacetCount, _
                                 adVertexCoords, adNormalVectors, aiVertexIndices)

' Offset vertices by given distance along anti-normals
For i = 0 To (iVertexCount * 3 - 1)
    adVertexCoords(i) = adVertexCoords(i) - adNormalVectors(i) * offset
Next

' Start a transaction.
Dim oTrans As Transaction
oTrans = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Z Height Colors")

' Create the graphics data sets collection.
oDataSets = oPartDoc.GraphicsDataSetsCollection.Add("CG_Test")

' Create the coordinate set and set it using the coordinates from the facets.
Dim oGraphicsCoordSet As GraphicsCoordinateSet
oGraphicsCoordSet = oDataSets.CreateCoordinateSet(1)
Call oGraphicsCoordSet.PutCoordinates(adVertexCoords)

' Create the index set and set it using the indices from the facets.
Dim oGraphicsIndexSet As GraphicsIndexSet
oGraphicsIndexSet = oDataSets.CreateIndexSet(2)
Call oGraphicsIndexSet.PutIndices(aiVertexIndices)

' Create the normal set and set it using the normals from the facets.
Dim oGraphicsNormalSet As GraphicsNormalSet
oGraphicsNormalSet = oDataSets.CreateNormalSet(3)
Call oGraphicsNormalSet.PutNormals(adNormalVectors)

' Allocate the array that will contain the color information.
' This array contains RGB values for each vertex.
Dim abtColors(0 To iVertexCount * 3 - 1) As Byte

' Load the array with color information for each vertex.
For i = 0 To iVertexCount - 1
    ' Set the color information for the current vertex.
    ' currently, all vertices are a constant color, but ...
    abtColors(i * 3) = 200
    abtColors(i * 3 + 1) = 0
    abtColors(i * 3 + 2) = 0
Next

' Create the color set and set it using the array of rgb values just created.
Dim oGraphicsColorSet As GraphicsColorSet
oGraphicsColorSet = oDataSets.CreateColorSet(4)
Call oGraphicsColorSet.PutColors(abtColors)

' Create a scalar data texture coordinate set representing distance from origin
Dim oTCSet As GraphicsTextureCoordinateSet
oTCSet = oDataSets.CreateTextureCoordinateSet(5)
Dim tc(0 To iVertexCount-1) As Double
Dim tcMax As Double
tcMax = 0
Dim tcMin As Double
tcMin = 1000000#
For i = 0 To iVertexCount - 1
    tc(i) = Math.Sqrt(adVertexCoords(3 * (i)) * adVertexCoords(3 * (i)) + adVertexCoords(3 * (i) + 1) * adVertexCoords(3 *
    tcMin = min(tcMin, tc(i))
    tcMax = max(tcMax, tc(i))
Next
Call oTCSet.PutCoordinates(tc)

' Create the client graphics collection.
Dim oClientGraphics As ClientGraphics
oClientGraphics = oPartDoc.ComponentDefinition.ClientGraphicsCollection.Add("CG_Test")

' Create a graphics node.
Dim oGraphicNode As GraphicsNode
oGraphicNode = oClientGraphics.AddNode(100)

' Create the triangle graphics.
Dim oTriangles As TriangleGraphics
oTriangles = oGraphicNode.AddTriangleGraphics

'=============================================

Dim oColorMapper As GraphicsColorMapper
oColorMapper = oDataSets.CreateColorMapper

'=============================================

'construct ColorMapper:
'  black - red - orange - yellow - green - cyan - blue - magenta - black
'      minV                        ...                         maxV
Dim nColors As Integer
nColors = 9

Dim cmColors(0 To nColors * 3 -1) As Byte
For i = 0 To nColors * 3 - 1
    cmColors(i) = 0
Next
'black
cmColors(3) = 255       'red
cmColors(6) = 255       'orange
cmColors(6 + 1) = 125
cmColors(9) = 255      'yellow
cmColors(9 + 1) = 255
cmColors(12 + 1) = 255  'green
cmColors(15 + 1) = 255  'cyan
cmColors(15 + 2) = 255
cmColors(18 + 2) = 255  'blue
cmColors(21) = 255      'magenta
cmColors(21 + 2) = 255
```

```
            'black

            Dim nValues As Integer
            If (mapperType = 0) Then
                nValues = nColors - 1   'discrete value boundaries
            Else
                nValues = nColors       'continuous value points
            End If

            Dim cmValues(0 To nValues - 1) As Double
            For i = 0 To nValues - 1
                cmValues(i) = tcMin + (tcMax - tcMin) * i / (nValues - 1#)
            Next

            '=========================================

            Call oColorMapper.PutColors(cmColors)
            Call oColorMapper.PutValues(cmValues)


            ' Set various properties of the triangle graphics.
            oTriangles.CoordinateSet = oGraphicsCoordSet
            oTriangles.CoordinateIndexSet = oGraphicsIndexSet
            oTriangles.NormalSet = oGraphicsNormalSet
            oTriangles.NormalBinding = kPerVertexNormals
            oTriangles.NormalIndexSet = oGraphicsIndexSet
        '    oTriangles.ColorSet = oGraphicsColorSet
        '    oTriangles.ColorBinding = kPerVertexColors
        '    oTriangles.ColorIndexSet = oGraphicsIndexSet
            oTriangles.TextureCoordinateSet = oTCSet
            oTriangles.TextureCoordinateIndexSet = oGraphicsIndexSet
            oTriangles.ColorMapper = oColorMapper

            ' End the transaction.
            oTrans.End

            ' Update the view.
            ThisApplication.ActiveView.Update
        End Sub

        Public Sub increaseOffset()
            lastOffset = lastOffset * 1.1
            Dim offset As Double
            offset = lastOffset
            Call OffsetSurfaceBy(offset)
        End Sub

        Public Sub DecreaseOffset()
            lastOffset = lastOffset * 0.9
            Dim offset As Double
            offset = lastOffset
            Call OffsetSurfaceBy(offset)
        End Sub

        Public Sub OffsetSurface()
            Dim oPartDoc As PartDocument
            oPartDoc = ThisApplication.ActiveDocument

            Dim lenunits As UnitsTypeEnum
            lenunits = oPartDoc.UnitsOfMeasure.LengthUnits
            Dim unitscale As Double
            Dim unitname As String
            If lenunits = kInchLengthUnits Then
                unitscale = 2.54
                unitname = "inches"
            Else
            If lenunits = kMillimeterLengthUnits Then
                unitscale = 0.1
                unitname = "millimeters"
            Else
            If lenunits = kCentimeterLengthUnits Then
                unitscale = 1#
                unitname = "centimeters"
            End If
            End If
            End If

            Dim offset As Double
            offset = 0
            On Error Resume Next
            offset = InputBox("Enter offset (in " + unitname + "):", "Offset", lastOffset / unitscale)
            On Error GoTo 0

            lastOffset = offset * unitscale

            Call OffsetSurfaceBy(lastOffset)
        End Sub
End Class
```

Client Graphics - Vertex Color by Z Height

Description

This sample demonstrates using client graphics and some other functions that help to support display control. It uses the currently active part and replaces the part display with a display where the part's color varies from blue to red where blue is assigned to the lowest Z portion of the part and red is assigned to the highest Z portion of the part. Areas in between are represented by a smooth blend of color from blue to red.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ZHeightColors()
    ' Get the surface body from the active document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    Dim oSurfBody As SurfaceBody
    Set oSurfBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)
    Set oSurfBody = oPartDoc.ComponentDefinitions.Item(1).SurfaceBodies.Item(1)

    ' Delete the graphics data set and client graphics, if they exist.
    Dim oDataSets As GraphicsDataSets
    On Error Resume Next
    Set oDataSets = oPartDoc.GraphicsDataSetsCollection.Item("MyTest")
    If Err.Number = 0 Then
        oDataSets.Delete
        oPartDoc.ComponentDefinition.ClientGraphicsCollection.Item("MyTest").Delete
        oSurfBody.Visible = True
        ThisApplication.ActiveView.Update
        Exit Sub
    End If
    On Error GoTo 0

    ' Determine the highest tolerance of the existing facet sets.
    Dim ToleranceCount As Long
    Dim ExistingTolerances() As Double
    Call oSurfBody.GetExistingFacetTolerances(ToleranceCount, ExistingTolerances)

    Dim i As Long
    Dim BestTolerance As Double
    For i = 0 To ToleranceCount - 1
        If i = 0 Then
            BestTolerance = ExistingTolerances(i)
        ElseIf ExistingTolerances(i) < BestTolerance Then
            BestTolerance = ExistingTolerances(i)
        End If
    Next

    ' Get a set of existing facets.
    Dim iVertexCount As Long
    Dim iFacetCount As Long
    Dim adVertexCoords() As Double
    Dim adNormalVectors() As Double
    Dim aiVertexIndices() As Long
    Call oSurfBody.GetExistingFacets(BestTolerance, iVertexCount, iFacetCount, _
                                     adVertexCoords, adNormalVectors, aiVertexIndices)

    ' Start a transaction.
    Dim oTrans As Transaction
    Set oTrans = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Z Height Colors")

    ' Create the graphics data sets collection.
    Set oDataSets = oPartDoc.GraphicsDataSetsCollection.Add("MyTest")

    ' Create the coordinate set and set it using the coordinates from the facets.
    Dim oGraphicsCoordSet As GraphicsCoordinateSet
    Set oGraphicsCoordSet = oDataSets.CreateCoordinateSet(1)
    Call oGraphicsCoordSet.PutCoordinates(adVertexCoords)

    ' Create the index set and set it using the indices from the facets.
    Dim oGraphicsIndexSet As GraphicsIndexSet
    Set oGraphicsIndexSet = oDataSets.CreateIndexSet(2)
    Call oGraphicsIndexSet.PutIndices(aiVertexIndices)

    ' Create the normal set and set it using the normals from the facets.
    Dim oGraphicsNormalSet As GraphicsNormalSet
    Set oGraphicsNormalSet = oDataSets.CreateNormalSet(3)
    Call oGraphicsNormalSet.PutNormals(adNormalVectors)

    ' Determine the min-max range of the body in Z.
    Dim dMinZ As Double
    dMinZ = oSurfBody.RangeBox.MinPoint.Z
    Dim dMaxZ As Double
    dMaxZ = oSurfBody.RangeBox.MaxPoint.Z
    Dim dHeightDifference As Double
    dHeightDifference = dMaxZ - dMinZ

    ' Allocate the array that will contain the color information.
    ' This array contains RGB values for each vertex.
    Dim abtColors() As Byte
    ReDim abtColors(0 To iVertexCount * 3 - 1) As Byte

    ' Load the array with color information for each vertex.
    For i = 0 To iVertexCount - 1
        ' Get the Z height of the current vertex.
        Dim dZValue As Double
        dZValue = adVertexCoords(i * 3 + 2)

        ' Set the color information for the current vertex.  It's computed by
        ' determining the percentage of the total Z range of the body this vertex
        ' is within.  A color between red and blue is computed based on this percentage.
        ' Blue is at the minimum Z and Red is at the maximum Z with blending between.
        abtColors(i * 3) = ((dZValue - dMinZ) / dHeightDifference) * 255
        abtColors(i * 3 + 1) = 0
        abtColors(i * 3 + 2) = ((dMaxZ - dZValue) / dHeightDifference) * 255
    Next

    ' Create the color set and set it using the array of rgb values just created.
    Dim oGraphicsColorSet As GraphicsColorSet
    Set oGraphicsColorSet = oDataSets.CreateColorSet(4)
    Call oGraphicsColorSet.PutColors(abtColors)

    ' Create the client graphics collection.
```

```
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oPartDoc.ComponentDefinition.ClientGraphicsCollection.Add("MyTest")

    ' Create a graphics node.
    Dim oGraphicNode As GraphicsNode
    Set oGraphicNode = oClientGraphics.AddNode(1)

    ' Create the triangle graphics.
    Dim oTriangles As TriangleGraphics
    Set oTriangles = oGraphicNode.AddTriangleGraphics

    ' Set various prroperties of the triangle graphics.
    oTriangles.CoordinateSet = oGraphicsCoordSet
    oTriangles.CoordinateIndexSet = oGraphicsIndexSet
    oTriangles.NormalSet = oGraphicsNormalSet
    oTriangles.NormalBinding = kPerVertexNormals
    oTriangles.NormalIndexSet = oGraphicsIndexSet
    oTriangles.ColorSet = oGraphicsColorSet
    oTriangles.ColorBinding = kPerVertexColors
    oTriangles.ColorIndexSet = oGraphicsIndexSet

    ' Turn off the display of the body.
    oSurfBody.Visible = False

    ' End the transaction.
    oTrans.End

    ' Update the view.
    ThisApplication.ActiveView.Update
End Sub
```

Copy Code

```
    ' Get the surface body from the active document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    Dim oSurfBody As SurfaceBody
    oSurfBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)
    oSurfBody = oPartDoc.ComponentDefinitions.Item(1).SurfaceBodies.Item(1)

    ' Delete the graphics data set and client graphics, if they exist.
    Dim oDataSets As GraphicsDataSets
    On Error Resume Next
    oDataSets = oPartDoc.GraphicsDataSetsCollection.Item("MyTest")
    If Err.Number = 0 Then
        oDataSets.Delete
        oPartDoc.ComponentDefinition.ClientGraphicsCollection.Item("MyTest").Delete
        oSurfBody.Visible = True
        ThisApplication.ActiveView.Update
        Exit Sub
    End If
    On Error GoTo 0

    ' Determine the highest tolerance of the existing facet sets.
    Dim ToleranceCount As Long
    Dim ExistingTolerances() As Double = New Double() {}
    Call oSurfBody.GetExistingFacetTolerances(ToleranceCount, ExistingTolerances)

    Dim i As Long
    Dim BestTolerance As Double
    For i = 0 To ToleranceCount - 1
        If i = 0 Then
            BestTolerance = ExistingTolerances(i)
        ElseIf ExistingTolerances(i) < BestTolerance Then
            BestTolerance = ExistingTolerances(i)
        End If
    Next

    ' Get a set of existing facets.
    Dim iVertexCount As Integer
    Dim iFacetCount As Integer
    Dim adVertexCoords() As Double = New Double() {}
    Dim adNormalVectors() As Double = New Double() {}
    Dim aiVertexIndices() As Integer = New Integer() {}
    Call oSurfBody.GetExistingFacets(BestTolerance, iVertexCount, iFacetCount, _
                                     adVertexCoords, adNormalVectors, aiVertexIndices)

    ' Start a transaction.
    Dim oTrans As Transaction
    oTrans = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Z Height Colors")

    ' Create the graphics data sets collection.
    oDataSets = oPartDoc.GraphicsDataSetsCollection.Add("MyTest")

    ' Create the coordinate set and set it using the coordinates from the facets.
    Dim oGraphicsCoordSet As GraphicsCoordinateSet
    oGraphicsCoordSet = oDataSets.CreateCoordinateSet(1)
    Call oGraphicsCoordSet.PutCoordinates(adVertexCoords)

    ' Create the index set and set it using the indices from the facets.
    Dim oGraphicsIndexSet As GraphicsIndexSet
    oGraphicsIndexSet = oDataSets.CreateIndexSet(2)
    Call oGraphicsIndexSet.PutIndices(aiVertexIndices)

    ' Create the normal set and set it using the normals from the facets.
    Dim oGraphicsNormalSet As GraphicsNormalSet
    oGraphicsNormalSet = oDataSets.CreateNormalSet(3)
    Call oGraphicsNormalSet.PutNormals(adNormalVectors)

    ' Determine the min-max range of the body in Z.
    Dim dMinZ As Double
    dMinZ = oSurfBody.RangeBox.MinPoint.Z
    Dim dMaxZ As Double
    dMaxZ = oSurfBody.RangeBox.MaxPoint.Z
    Dim dHeightDifference As Double
    dHeightDifference = dMaxZ - dMinZ
```

```
    ' Allocate the array that will contain the color information.
    ' This array contains RGB values for each vertex.
    Dim abtColors(0 To iVertexCount * 3 - 1) As Byte

    ' Load the array with color information for each vertex.
    For i = 0 To iVertexCount - 1
        ' Get the Z height of the current vertex.
        Dim dZValue As Double
        dZValue = adVertexCoords(i * 3 + 2)

        ' Set the color information for the current vertex.  It's computed by
        ' determining the percentage of the total Z range of the body this vertex
        ' is within.  A color between red and blue is computed based on this percentage.
        ' Blue is at the minimum Z and Red is at the maximum Z with blending between.
        abtColors(i * 3) = ((dZValue - dMinZ) / dHeightDifference) * 255
        abtColors(i * 3 + 1) = 0
        abtColors(i * 3 + 2) = ((dMaxZ - dZValue) / dHeightDifference) * 255
    Next

    ' Create the color set and set it using the array of rgb values just created.
    Dim oGraphicsColorSet As GraphicsColorSet
    oGraphicsColorSet = oDataSets.CreateColorSet(4)
    Call oGraphicsColorSet.PutColors(abtColors)

    ' Create the client graphics collection.
    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oPartDoc.ComponentDefinition.ClientGraphicsCollection.Add("MyTest")

    ' Create a graphics node.
    Dim oGraphicNode As GraphicsNode
    oGraphicNode = oClientGraphics.AddNode(1)

    ' Create the triangle graphics.
    Dim oTriangles As TriangleGraphics
    oTriangles = oGraphicNode.AddTriangleGraphics

    ' Set various prroperties of the triangle graphics.
    oTriangles.CoordinateSet = oGraphicsCoordSet
    oTriangles.CoordinateIndexSet = oGraphicsIndexSet
    oTriangles.NormalSet = oGraphicsNormalSet
    oTriangles.NormalBinding = kPerVertexNormals
    oTriangles.NormalIndexSet = oGraphicsIndexSet
    oTriangles.ColorSet = oGraphicsColorSet
    oTriangles.ColorBinding = kPerVertexColors
    oTriangles.ColorIndexSet = oGraphicsIndexSet

    ' Turn off the display of the body.
    oSurfBody.Visible = False

    ' End the transaction.
    oTrans.End

    ' Update the view.
    ThisApplication.ActiveView.Update
```

Client Graphics - Line

Description

This sample demonstrates the creation of custom graphics using LineGraphics and LineStripGraphics. The same set of coordinate data is used for both types of graphics. Line graphics use two coordinates to define a line, and then the next two coordinates to define the next line, and so on through the defined coordinates. For the data provided, this results in gaps in the drawn curve. Line strips use the first two coordinates to define the first line and then the last point of the first line becomes the first point of the second line and the next coordinate is used as the end point of the second line. This results in the set of points being connected by a continuous set of lines, drawing a continuous curve. This sample also demonstrates two methods of defining the color for client graphics. In one case it uses an existing appearance asset, and in the other, it explicitly defines a color and assigns it. To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DrawCustomLines()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document.  If it doesn't create it.
    On Error Resume Next
    Dim oGraphicsData As GraphicsDataSets
    Set oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oGraphicsData.Delete
        oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID").Delete

        ' update the display to see the results.
```

```
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        Set oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        Set oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Create an array that contains coordinates that define a set
        ' of outwardly spiraling points.
        Dim oPointCoords(1 To 90) As Double
        Dim i As Long
        Dim dRadius As Double
        dRadius = 1
        Dim dAngle As Double
        For i = 0 To 29
            ' Define the X, Y, and Z components of the point.
            oPointCoords(i * 3 + 1) = dRadius * Cos(dAngle)
            oPointCoords(i * 3 + 2) = dRadius * Sin(dAngle)
            oPointCoords(i * 3 + 3) = i / 2

            ' Increment the angle and radius to create the spiral.
            dRadius = dRadius + 0.25
            dAngle = dAngle + (3.14159265358979 / 6)
        Next

        ' Assign the points into the coordinate set.
        Call oCoordSet.PutCoordinates(oPointCoords)

        ' Create the ClientGraphics object.
        Dim oClientGraphics As ClientGraphics
        Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oLineNode As GraphicsNode
        Set oLineNode = oClientGraphics.AddNode(1)

        ' Create a LineGraphics object within the node.
        Dim oLineSet As LineGraphics
        Set oLineSet = oLineNode.AddLineGraphics

        ' Assign the coordinate set to the line graphics.
        oLineSet.CoordinateSet = oCoordSet

        ' Assign a color to the node using an existing appearance asset.
        Dim oAppearance As Asset
        Set oAppearance = oDoc.AppearanceAssets(1)

        oLineNode.Appearance = oAppearance

        ' Update the view to see the resulting spiral.
        ThisApplication.ActiveView.Update

        ' Create another graphics node for a line strip.
        Dim oLineStripNode As GraphicsNode
        Set oLineStripNode = oClientGraphics.AddNode(2)

        ' Create a LineStripGraphics object within the new node.
        Dim oLineStrip As LineStripGraphics
        Set oLineStrip = oLineStripNode.AddLineStripGraphics

        ' Assign the same coordinate set to the line strip.
        oLineStrip.CoordinateSet = oCoordSet

        ' Create a color set to use in defining a explicit color to the line strip.
        Dim oColorSet As GraphicsColorSet
        Set oColorSet = oDataSets.CreateColorSet(1)

        ' Add a single color to the set that is red.
        Call oColorSet.Add(1, 255, 0, 0)

        ' Assign the color set to the line strip.
        oLineStrip.ColorSet = oColorSet

        ' The two spirals are currently on top of each other so translate the
        ' new one in the x direction so they're side by side.
        Dim oMatrix As Matrix
        Set oMatrix = oLineStripNode.Transformation
        Call oMatrix.SetTranslation(oTransGeom.CreateVector(15, 0, 0))
        oLineStripNode.Transformation = oMatrix

        ' Update the view to see the resulting spiral.
        ThisApplication.ActiveView.Update
    End If
End Sub
```

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
```

```
' If it does clean up by removing all associated of the client graphics
' from the document.  If it doesn't create it.
On Error Resume Next
Dim oGraphicsData As GraphicsDataSets
oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
If Err.Number = 0 Then
    On Error GoTo 0
    ' An existing client graphics object was successfully obtained so clean up.
    oGraphicsData.Delete
    oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID").Delete

    ' update the display to see the results.
    ThisApplication.ActiveView.Update
Else
    Err.Clear
    On Error GoTo 0

    ' Set a reference to the transient geometry object for user later.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a graphics data set object.  This object contains all of the
    ' information used to define the graphics.
    Dim oDataSets As GraphicsDataSets
    oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

    ' Create a coordinate set.
    Dim oCoordSet As GraphicsCoordinateSet
    oCoordSet = oDataSets.CreateCoordinateSet(1)

    ' Create an array that contains coordinates that define a set
    ' of outwardly spiraling points.
    Dim oPointCoords(0 To 89) As Double
    Dim i As Long
    Dim dRadius As Double
    dRadius = 1
    Dim dAngle As Double
    For i = 0 To 29
        ' Define the X, Y, and Z components of the point.
        oPointCoords(i * 3 ) = dRadius * Cos(dAngle)
        oPointCoords(i * 3 + 1) = dRadius * Sin(dAngle)
        oPointCoords(i * 3 + 2) = i / 2

        ' Increment the angle and radius to create the spiral.
        dRadius = dRadius + 0.25
        dAngle = dAngle + (3.14159265358979 / 6)
    Next

    ' Assign the points into the coordinate set.
    Call oCoordSet.PutCoordinates(oPointCoords)

    ' Create the ClientGraphics object.
    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

    ' Create a new graphics node within the client graphics objects.
    Dim oLineNode As GraphicsNode
    oLineNode = oClientGraphics.AddNode(1)

    ' Create a LineGraphics object within the node.
    Dim oLineSet As LineGraphics
    oLineSet = oLineNode.AddLineGraphics

    ' Assign the coordinate set to the line graphics.
    oLineSet.CoordinateSet = oCoordSet

    ' Assign a color to the node using an existing appearance asset.
    Dim oAppearance As Asset
    oAppearance = oDoc.AppearanceAssets(1)

    oLineNode.Appearance = oAppearance

    ' Update the view to see the resulting spiral.
    ThisApplication.ActiveView.Update

    ' Create another graphics node for a line strip.
    Dim oLineStripNode As GraphicsNode
    oLineStripNode = oClientGraphics.AddNode(2)

    ' Create a LineStripGraphics object within the new node.
    Dim oLineStrip As LineStripGraphics
    oLineStrip = oLineStripNode.AddLineStripGraphics

    ' Assign the same coordinate set to the line strip.
    oLineStrip.CoordinateSet = oCoordSet

    ' Create a color set to use in defining a explicit color to the line strip.
    Dim oColorSet As GraphicsColorSet
    oColorSet = oDataSets.CreateColorSet(1)

    ' Add a single color to the set that is red.
    Call oColorSet.Add(1, 255, 0, 0)

    ' Assign the color set to the line strip.
    oLineStrip.ColorSet = oColorSet

    ' The two spirals are currently on top of each other so translate the
    ' new one in the x direction so they're side by side.
    Dim oMatrix As Matrix
    oMatrix = oLineStripNode.Transformation
    Call oMatrix.SetTranslation(oTransGeom.CreateVector(15, 0, 0))
    oLineStripNode.Transformation = oMatrix

    ' Update the view to see the resulting spiral.
    ThisApplication.ActiveView.Update
End If
```

Client graphics from SAT file body

Description

The following sample demonstrates how to display client graphics based on bodies read in from a SAT file.

Code Samples

- VBA
- iLogic

Make sure you have "C:\temp\block.sat" or change the path in the code below.

Copy Code

```
Public Sub ClientGraphicsFromSATFileBody()
    ' Set a reference to the TransientBRep object
    Dim oTransientBRep As TransientBRep
    Set oTransientBRep = ThisApplication.TransientBRep

    ' Get the first body from the specified sat file
    Dim oBody As SurfaceBody
    Set oBody = oTransientBRep.ReadFromFile("C:\temp\block.sat").Item(1)

    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create the ClientGraphics object.
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

    ' Create a new graphics node within the client graphics objects.
    Dim oSurfacesNode As GraphicsNode
    Set oSurfacesNode = oClientGraphics.AddNode(1)

    ' Create client graphics based on the transient body
    Dim oSurfaceGraphics As SurfaceGraphics
    Set oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)

    ' Update the view to see the resulting curves.
    ThisApplication.ActiveView.Update
End Sub
```

Make sure you have "C:\temp\block.sat" or change the path in the code below.

Copy Code

```
    ' Set a reference to the TransientBRep object
    Dim oTransientBRep As TransientBRep
    oTransientBRep = ThisApplication.TransientBRep

    ' Get the first body from the specified sat file
    Dim oBody As SurfaceBody
    oBody = oTransientBRep.ReadFromFile("C:\temp\block.sat").Item(1)

    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create the ClientGraphics object.
    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Sample3DGraphicsID")

    ' Create a new graphics node within the client graphics objects.
    Dim oSurfacesNode As GraphicsNode
    oSurfacesNode = oClientGraphics.AddNode(1)

    ' Create client graphics based on the transient body
    Dim oSurfaceGraphics As SurfaceGraphics
    oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)

    ' Update the view to see the resulting curves.
    ThisApplication.ActiveView.Update
```

Text Using Client Graphics (Simple)

Description

This sample demonstrates creating text using client graphics. It illustrates the simple case where the text is one font and is one line.

Code Samples

- VBA
- iLogic

To run the sample have a part or assembly document open.

Copy Code

```
Public Sub ClientGraphicsText()
    ' Set a reference to the document.  This will work with
    ' either a part or assembly document.
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    ' Attempt to get the existing client graphics object.  If it exists
    ' delete it so the rest of the code can continue as if it never existed.
    Dim oClientGraphics As ClientGraphics
    On Error Resume Next
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Text Test")
    If Err.Number = 0 Then
        oClientGraphics.Delete
    End If
    On Error GoTo 0
    ThisApplication.ActiveView.Update

    ' Create a new ClientGraphics object.
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Text Test")

    ' Create a graphics node.
    Dim oNode As GraphicsNode
    Set oNode = oClientGraphics.AddNode(1)

    ' Create text graphics.
    Dim oTextGraphics As TextGraphics
    Set oTextGraphics = oNode.AddTextGraphics

    ' Set the properties of the text.
    oTextGraphics.Text = "This is the sample text."
    oTextGraphics.Anchor = ThisApplication.TransientGeometry.CreatePoint(0, 0, 0)
    oTextGraphics.Bold = True
    oTextGraphics.Font = "Arial"
    oTextGraphics.FontSize = 40
    oTextGraphics.HorizontalAlignment = kAlignTextLeft
    oTextGraphics.Italic = True
    Call oTextGraphics.PutTextColor(0, 255, 0)
    oTextGraphics.VerticalAlignment = kAlignTextMiddle

    ' Update the view to see the text.
    ThisApplication.ActiveView.Update
End Sub
```

To run the sample have a part or assembly document open.

Copy Code

```
    ' Set a reference to the document.  This will work with
    ' either a part or assembly document.
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As ComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    ' Attempt to get the existing client graphics object.  If it exists
    ' delete it so the rest of the code can continue as if it never existed.
    Dim oClientGraphics As ClientGraphics
    On Error Resume Next
    oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Text Test")
    If Err.Number = 0 Then
        oClientGraphics.Delete
    End If
    On Error GoTo 0
    ThisApplication.ActiveView.Update

    ' Create a new ClientGraphics object.
    oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Text Test")

    ' Create a graphics node.
    Dim oNode As GraphicsNode
    oNode = oClientGraphics.AddNode(1)

    ' Create text graphics.
    Dim oTextGraphics As TextGraphics
    oTextGraphics = oNode.AddTextGraphics

    ' Set the properties of the text.
    oTextGraphics.Text = "This is the sample text."
    oTextGraphics.Anchor = ThisApplication.TransientGeometry.CreatePoint(0, 0, 0)
    oTextGraphics.Bold = True
    oTextGraphics.Font = "Arial"
    oTextGraphics.FontSize = 40
    oTextGraphics.HorizontalAlignment = kAlignTextLeft
    oTextGraphics.Italic = True
    Call oTextGraphics.PutTextColor(0, 255, 0)
    oTextGraphics.VerticalAlignment = kAlignTextMiddle

    ' Update the view to see the text.
    ThisApplication.ActiveView.Update
```

Text Using Client Graphics (Multiple fonts and lines)

Description

This sample demonstrates creating text using client graphics. It illustrates the more complex case of changes in font and more than one line.

Code Samples

- VBA
- iLogic

To run the sample have a part or assembly document open.

Copy Code

```
Public Sub ClientGraphicsTextMultipleFonts()
    ' Set a reference to the document.  This will work with
    ' either a part or assembly document.
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    ' Attempt to get the existing client graphics object.  If it exists
    ' delete it so the rest of the code can continue as if it never existed.
    Dim oClientGraphics As ClientGraphics
    On Error Resume Next
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Text Test")
    If Err.Number = 0 Then
        oClientGraphics.Delete
    End If
    On Error GoTo 0
    ThisApplication.ActiveView.Update

    ' Create a new ClientGraphics object.
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Text Test")

    ' Create a graphics node.
    Dim oNode As GraphicsNode
    Set oNode = oClientGraphics.AddNode(1)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oModelAnchorPoint As Point
    Set oModelAnchorPoint = oTG.CreatePoint(50, 0, 0)

    ' Create several text graphics objects, one for each font change.  The anchor of the
    ' TextGraphics object defines the position of each text element relative to each other.
    ' Because they're all drawn with pixel scaling behavior these coordinates are in
    ' pixel space.  They all use the same point as input for the SetTransformBehavior call.
    ' This point is in model space and defines their anchor within the model.

    ' Draw the first character which is the diameter symbol.
    Dim oTextGraphics(1 To 4) As TextGraphics
    Set oTextGraphics(1) = oNode.AddTextGraphics
    oTextGraphics(1).Text = "n "

    ' Because this text will have pixel scaling behavior these coordinates are in pixel space.
    oTextGraphics(1).Anchor = oTG.CreatePoint(0, 0, 0)

    oTextGraphics(1).Font = "AIGDT"
    oTextGraphics(1).FontSize = 25
    oTextGraphics(1).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(1).PutTextColor(0, 255, 255)
    oTextGraphics(1).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(1).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)
    Dim oBox As Box
    Set oBox = oTextGraphics(1).RangeBox

    ' Draw the next section of the string relative to the first section.
    Set oTextGraphics(2) = oNode.AddTextGraphics
    oTextGraphics(2).Text = "9.4 - 9.8"

    ' The range of the previous character is used to determine where to position
    ' the next string.  The range is returned in pixels.
    oTextGraphics(2).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MaxPoint.X, 0, 0)

    oTextGraphics(2).Font = "Arial"
    oTextGraphics(2).FontSize = 25
    oTextGraphics(2).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(2).PutTextColor(0, 255, 255)
    oTextGraphics(2).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(2).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)

    ' Draw a depth symbol on the next line.
    Set oTextGraphics(3) = oNode.AddTextGraphics
    oTextGraphics(3).Text = "x "

    ' Using the range of the first text graphics determine the position for a string that
    ' is immediately below.  This defines the next line of text.
    oTextGraphics(3).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MinPoint.X, oBox.MinPoint.Y - ((oBox.MaxPoint.Y - oBox.MinPoi
    oTextGraphics(3).Font = "AIGDT"
    oTextGraphics(3).FontSize = 25
    oTextGraphics(3).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(3).PutTextColor(0, 255, 255)
```

```
    oTextGraphics(3).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(3).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)
    Set oBox = oTextGraphics(3).RangeBox

     ' Draw the last set of text.s
    Set oTextGraphics(4) = oNode.AddTextGraphics
    oTextGraphics(4).Text = "20"
    oTextGraphics(4).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MaxPoint.X, oTextGraphics(3).Anchor.Y, 0)
    oTextGraphics(4).Font = "Arial"
    oTextGraphics(4).FontSize = 25
    oTextGraphics(4).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(4).PutTextColor(0, 255, 255)
    oTextGraphics(4).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(4).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)

    ' Update the view to see the text.
    ThisApplication.ActiveView.Update
End Sub
```

To run the sample have a part or assembly document open.

Copy Code

```
    ' Set a reference to the document.  This will work with
    ' either a part or assembly document.
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As ComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    ' Attempt to get the existing client graphics object.  If it exists
    ' delete it so the rest of the code can continue as if it never existed.
    Dim oClientGraphics As ClientGraphics
    On Error Resume Next
    oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Text Test")
    If Err.Number = 0 Then
        oClientGraphics.Delete
    End If
    On Error GoTo 0
    ThisApplication.ActiveView.Update

    ' Create a new ClientGraphics object.
    oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Text Test")

    ' Create a graphics node.
    Dim oNode As GraphicsNode
    oNode = oClientGraphics.AddNode(1)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oModelAnchorPoint As Point
    oModelAnchorPoint = oTG.CreatePoint(50, 0, 0)

    ' Create several text graphics objects, one for each font change.  The anchor of the
    ' TextGraphics object defines the position of each text element relative to each other.
    ' Because they're all drawn with pixel scaling behavior these coordinates are in
    ' pixel space.  They all use the same point as input for the SetTransformBehavior call.
    ' This point is in model space and defines their anchor within the model.

    ' Draw the first character which is the diameter symbol.
    Dim oTextGraphics(0 To 3) As TextGraphics
    oTextGraphics(0) = oNode.AddTextGraphics
    oTextGraphics(0).Text = "n "

    ' Because this text will have pixel scaling behavior these coordinates are in pixel space.
    oTextGraphics(0).Anchor = oTG.CreatePoint(0, 0, 0)

    oTextGraphics(0).Font = "AIGDT"
    oTextGraphics(0).FontSize = 25
    oTextGraphics(0).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(0).PutTextColor(0, 255, 255)
    oTextGraphics(0).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(0).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)
    Dim oBox As Box
    oBox = oTextGraphics(0).RangeBox

    ' Draw the next section of the string relative to the first section.
    oTextGraphics(1) = oNode.AddTextGraphics
    oTextGraphics(1).Text = "9.4 - 9.8"

    ' The range of the previous character is used to determine where to position
    ' the next string.  The range is returned in pixels.
    oTextGraphics(1).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MaxPoint.X, 0, 0)

    oTextGraphics(1).Font = "Arial"
    oTextGraphics(1).FontSize = 25
    oTextGraphics(1).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(1).PutTextColor(0, 255, 255)
    oTextGraphics(1).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(1).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)

    ' Draw a depth symbol on the next line.
    oTextGraphics(2) = oNode.AddTextGraphics
    oTextGraphics(2).Text = "x "

    ' Using the range of the first text graphics determine the position for a string that
    ' is immediately below.  This defines the next line of text.
    oTextGraphics(2).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MinPoint.X, oBox.MinPoint.Y - ((oBox.MaxPoint.Y - oBox.MinPoi
    oTextGraphics(2).Font = "AIGDT"
    oTextGraphics(2).FontSize = 25
    oTextGraphics(2).HorizontalAlignment = kAlignTextLeft
    Call oTextGraphics(2).PutTextColor(0, 255, 255)
    oTextGraphics(2).VerticalAlignment = kAlignTextMiddle
    Call oTextGraphics(2).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)
    oBox = oTextGraphics(2).RangeBox
```

```
 ' Draw the last set of text.s
oTextGraphics(3) = oNode.AddTextGraphics
oTextGraphics(3).Text = "20"
oTextGraphics(3).Anchor = oTG.CreatePoint(oModelAnchorPoint.X + oBox.MaxPoint.X, oTextGraphics(2).Anchor.Y, 0)
oTextGraphics(3).Font = "Arial"
oTextGraphics(3).FontSize = 25
oTextGraphics(3).HorizontalAlignment = kAlignTextLeft
Call oTextGraphics(3).PutTextColor(0, 255, 255)
oTextGraphics(3).VerticalAlignment = kAlignTextMiddle
Call oTextGraphics(3).SetTransformBehavior(oModelAnchorPoint, kFrontFacingAndPixelScaling)

' Update the view to see the text.
ThisApplication.ActiveView.Update
```

Client Graphics - Triangle

Description

This sample demonstrates the creation of client graphics triangles using triange fans and strips. It does this by drawing a cylinder. The end caps are triangle fans and the cylinder is made from a triangle strip.

Code Samples

- VBA
- iLogic

To run this sample you need to have an assembly or part document open. The sample code will prompt for the number of sides of the cylinder. This is the number of facets the cylinder will be approximated by.

Copy Code

```
Public Sub DrawCustomTriangles()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document.  If it doesn't create it and draw a line.
    On Error Resume Next
    Dim oGraphicsData As GraphicsDataSets
    Set oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oGraphicsData.Delete
        oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID").Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        Set oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        Set oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Get the number of sides for the cylinder.
        Dim iSideCount As Long
        iSideCount = InputBox("Enter number of sides for the cylinder.  (Must be greater than 2.)", "Cylinder Tolerance", 25)

        Dim dRadius As Double
        Dim dHeight As Double
        dRadius = 3
        dHeight = 7

        ' Create an array containing points to used in drawing the cylinder.
        ' The points could also be directly defined in a coordinate set, but
        ' defining them in array and then setting the coordinates using the
        ' array is more efficient for a large set of points.
        Dim adPointCoords() As Double
        ReDim adPointCoords(1 To ((iSideCount + 1) * 2) * 3) As Double
        Dim i As Long
        Dim dAngle As Double

        ' Define the points for the outline of the first end of the cylinder.
        dAngle = 0
        For i = 0 To iSideCount - 1
            adPointCoords(i * 3 + 1) = dRadius * Cos(dAngle)
            adPointCoords(i * 3 + 2) = dRadius * Sin(dAngle)
            adPointCoords(i * 3 + 3) = 0

            ' Increment the angle for the next point
```

```
    dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
Next

' Define the points for the outline of the other end of the cylinder.
dAngle = 0
For i = iSideCount To (iSideCount * 2) - 1
    adPointCoords(i * 3 + 1) = dRadius * Cos(dAngle)
    adPointCoords(i * 3 + 2) = dRadius * Sin(dAngle)
    adPointCoords(i * 3 + 3) = dHeight

    ' Increment the angle for the next point
    dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
Next

' Define coordinate at the center of the first end of the cylinder.
adPointCoords((iSideCount * 2) * 3 + 1) = 0
adPointCoords((iSideCount * 2) * 3 + 2) = 0
adPointCoords((iSideCount * 2) * 3 + 3) = 0

' Define coordinate at the center of the other end of the cylinder.
adPointCoords((iSideCount * 2) * 3 + 4) = 0
adPointCoords((iSideCount * 2) * 3 + 5) = 0
adPointCoords((iSideCount * 2) * 3 + 6) = dHeight

' Assign the points to the coordinate set.
Call oCoordSet.PutCoordinates(adPointCoords)

' Create an index set to use for the triangle fan for
' cap of the first end of the cylinder.  This will
' serve as in index look-up into the coordinate set.
Dim oCap1Index As GraphicsIndexSet
Set oCap1Index = oDataSets.CreateIndexSet(1)

' Set the index values.  This could also be done by setting the
' value in an array and then using the array to set the values
' of the index set.  Using an array is more efficient, but this
' is used to here to demonstrate the IndexSet object's Add method.
Call oCap1Index.Add(1, iSideCount * 2 + 1)
Call oCap1Index.Add(2, 1)
Call oCap1Index.Add(3, 2)
For i = 4 To iSideCount + 1
    Call oCap1Index.Add(i, i - 1)
Next
Call oCap1Index.Add(iSideCount + 2, 1)

' Create an index set to use for the triangle fan of the other cylinder cap.
Dim oCap2Index As GraphicsIndexSet
Set oCap2Index = oDataSets.CreateIndexSet(2)

' Set the index values.
Call oCap2Index.Add(1, iSideCount * 2 + 2)
Call oCap2Index.Add(2, iSideCount + 1)
Call oCap2Index.Add(3, iSideCount + 2)
For i = 4 To iSideCount + 1
    Call oCap2Index.Add(i, i + iSideCount - 1)
Next
Call oCap2Index.Add(iSideCount + 2, iSideCount + 1)

' Create an index set to use for the triangle strip that will
' define the sides of the cylinder.
Dim oCylinderIndex As GraphicsIndexSet
Set oCylinderIndex = oDataSets.CreateIndexSet(3)

' Define the index values in an array.
Dim iIndexValues() As Long
ReDim iIndexValues(1 To (iSideCount + 1) * 2) As Long
For i = 0 To iSideCount - 1
    iIndexValues(i * 2 + 1) = i + 1
    iIndexValues(i * 2 + 2) = i + iSideCount + 1
Next
iIndexValues(((iSideCount + 1) * 2) - 1) = 1
iIndexValues((iSideCount + 1) * 2) = iSideCount + 1

' Define the index values in the index set using the array.
Call oCylinderIndex.PutIndices(iIndexValues)

' Create the ClientGraphics object.
Dim oClientGraphics As ClientGraphics
Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

' Create a new graphics node within the client graphics objects.
Dim oCylinderNode As GraphicsNode
Set oCylinderNode = oClientGraphics.AddNode(1)

' Create a triangle fan for cap 1 of the cylinder.
Dim oCap1TriangleFan As TriangleFanGraphics
Set oCap1TriangleFan = oCylinderNode.AddTriangleFanGraphics

' Set the coordinates and index set for the cap.
oCap1TriangleFan.CoordinateSet = oCoordSet
oCap1TriangleFan.CoordinateIndexSet = oCap1Index

' Create a triangle fan for cap 2 of the cylinder.
Dim oCap2TriangleFan As TriangleFanGraphics
Set oCap2TriangleFan = oCylinderNode.AddTriangleFanGraphics

' Set the coordinates and index set for the cap.
oCap2TriangleFan.CoordinateSet = oCoordSet
oCap2TriangleFan.CoordinateIndexSet = oCap2Index

' Create a triangle string for the sides of the cylinder.
Dim oCylinderStrip As TriangleStripGraphics
Set oCylinderStrip = oCylinderNode.AddTriangleStripGraphics

' Set the coordinates and index set for the cap.
oCylinderStrip.CoordinateSet = oCoordSet
oCylinderStrip.CoordinateIndexSet = oCylinderIndex
```

```
        Dim oAppearance As Asset
        Set oAppearance = oDoc.AppearanceAssets(1)

        oCylinderNode.Appearance = oAppearance

        ' update the display to see the results.
        ThisApplication.ActiveView.Update

        ' Define a normal for cap 1.
        Dim oCap1Normals As GraphicsNormalSet
        Set oCap1Normals = oDataSets.CreateNormalSet(1)
        Call oCap1Normals.Add(1, oTransGeom.CreateUnitVector(0, 0, -1))

        ' Assign the normals to the cap.
        oCap1TriangleFan.NormalSet = oCap1Normals

        ' Define a normal for cap 2.
        Dim oCap2Normals As GraphicsNormalSet
        Set oCap2Normals = oDataSets.CreateNormalSet(2)
        Call oCap2Normals.Add(1, oTransGeom.CreateUnitVector(0, 0, 1))

        ' Assign the normals to the cap.
        oCap2TriangleFan.NormalSet = oCap2Normals

        ' Create an array that contains the normals for each vertex of the cylinder.
        Dim adNormals() As Double
        ReDim adNormals(1 To ((iSideCount + 1) * 2) * 3) As Double
        dAngle = 0
        For i = 0 To iSideCount
            Dim dX As Double
            Dim dY As Double
            dX = Cos(dAngle)
            dY = Sin(dAngle)
            adNormals(i * 6 + 1) = dX
            adNormals(i * 6 + 2) = dY
            adNormals(i * 6 + 3) = 0
            adNormals(i * 6 + 4) = dX
            adNormals(i * 6 + 5) = dY
            adNormals(i * 6 + 6) = 0

            ' Increment the angle for the next normal.
            dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
        Next

        ' Create and set the normal set for the cylinder.
        Dim oCylinderNormals As GraphicsNormalSet
        Set oCylinderNormals = oDataSets.CreateNormalSet(3)
        Call oCylinderNormals.PutNormals(adNormals)

        ' Assign the normals to the cylinder.
        oCylinderStrip.NormalSet = oCylinderNormals

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    End If
End Sub
```

To run this sample you need to have an assembly or part document open. The sample code will prompt for the number of sides of the cylinder. This is the number of facets the cylinder will be approximated by.

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document.  If it doesn't create it and draw a line.
    On Error Resume Next
    Dim oGraphicsData As GraphicsDataSets
    oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oGraphicsData.Delete
        oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID").Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Get the number of sides for the cylinder.
        Dim iSideCount As Long
        iSideCount = InputBox("Enter number of sides for the cylinder.  (Must be greater than 2.)", "Cylinder Tolerance", 25)

        Dim dRadius As Double
```

```
Dim dHeight As Double
dRadius = 3
dHeight = 7

' Create an array containing points to used in drawing the cylinder.
' The points could also be directly defined in a coordinate set, but
' defining them in array and then setting the coordinates using the
' array is more efficient for a large set of points.
Dim adPointCoords(0 To ((iSideCount + 1) * 2) * 3 - 1) As Double
Dim i As Long
Dim dAngle As Double

' Define the points for the outline of the first end of the cylinder.
dAngle = 0
For i = 0 To iSideCount - 1
    adPointCoords(i * 3) = dRadius * Cos(dAngle)
    adPointCoords(i * 3 + 1) = dRadius * Sin(dAngle)
    adPointCoords(i * 3 + 2) = 0

    ' Increment the angle for the next point
    dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
Next

' Define the points for the outline of the other end of the cylinder.
dAngle = 0
For i = iSideCount To (iSideCount * 2) - 1
    adPointCoords(i * 3) = dRadius * Cos(dAngle)
    adPointCoords(i * 3 + 1) = dRadius * Sin(dAngle)
    adPointCoords(i * 3 + 2) = dHeight

    ' Increment the angle for the next point
    dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
Next

' Define coordinate at the center of the first end of the cylinder.
adPointCoords((iSideCount * 2) * 3) = 0
adPointCoords((iSideCount * 2) * 3 + 1) = 0
adPointCoords((iSideCount * 2) * 3 + 2) = 0

' Define coordinate at the center of the other end of the cylinder.
adPointCoords((iSideCount * 2) * 3 + 3) = 0
adPointCoords((iSideCount * 2) * 3 + 4) = 0
adPointCoords((iSideCount * 2) * 3 + 5) = dHeight

' Assign the points to the coordinate set.
Call oCoordSet.PutCoordinates(adPointCoords)

' Create an index set to use for the triangle fan for
' cap of the first end of the cylinder.  This will
' serve as in index look-up into the coordinate set.
Dim oCap1Index As GraphicsIndexSet
oCap1Index = oDataSets.CreateIndexSet(1)

' Set the index values.  This could also be done by setting the
' value in an array and then using the array to set the values
' of the index set.  Using an array is more efficient, but this
' is used to here to demonstrate the IndexSet object's Add method.
Call oCap1Index.Add(1, iSideCount * 2 + 1)
Call oCap1Index.Add(2, 1)
Call oCap1Index.Add(3, 2)
For i = 4 To iSideCount + 1
    Call oCap1Index.Add(i, i - 1)
Next
Call oCap1Index.Add(iSideCount + 2, 1)

' Create an index set to use for the triangle fan of the other cylinder cap.
Dim oCap2Index As GraphicsIndexSet
oCap2Index = oDataSets.CreateIndexSet(2)

' Set the index values.
Call oCap2Index.Add(1, iSideCount * 2 + 2)
Call oCap2Index.Add(2, iSideCount + 1)
Call oCap2Index.Add(3, iSideCount + 2)
For i = 4 To iSideCount + 1
    Call oCap2Index.Add(i, i + iSideCount - 1)
Next
Call oCap2Index.Add(iSideCount + 2, iSideCount + 1)

' Create an index set to use for the triangle strip that will
' define the sides of the cylinder.
Dim oCylinderIndex As GraphicsIndexSet
oCylinderIndex = oDataSets.CreateIndexSet(3)

' Define the index values in an array.
Dim iIndexValues(0 To (iSideCount + 1) * 2 - 1) As Integer
For i = 0 To iSideCount - 1
    iIndexValues(i * 2) = i + 1
    iIndexValues(i * 2 + 1) = i + iSideCount + 1
Next
iIndexValues(iSideCount * 2) = 1
iIndexValues(iSideCount * 2 + 1) = iSideCount + 1

' Define the index values in the index set using the array.
Call oCylinderIndex.PutIndices(iIndexValues)

' Create the ClientGraphics object.
Dim oClientGraphics As ClientGraphics
oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

' Create a new graphics node within the client graphics objects.
Dim oCylinderNode As GraphicsNode
oCylinderNode = oClientGraphics.AddNode(1)

' Create a triangle fan for cap 1 of the cylinder.
Dim oCap1TriangleFan As TriangleFanGraphics
oCap1TriangleFan = oCylinderNode.AddTriangleFanGraphics
```

```
        ' Set the coordinates and index set for the cap.
        oCap1TriangleFan.CoordinateSet = oCoordSet
        oCap1TriangleFan.CoordinateIndexSet = oCap1Index

        ' Create a triangle fan for cap 2 of the cylinder.
        Dim oCap2TriangleFan As TriangleFanGraphics
        oCap2TriangleFan = oCylinderNode.AddTriangleFanGraphics

        ' Set the coordinates and index set for the cap.
        oCap2TriangleFan.CoordinateSet = oCoordSet
        oCap2TriangleFan.CoordinateIndexSet = oCap2Index

        ' Create a triangle string for the sides of the cylinder.
        Dim oCylinderStrip As TriangleStripGraphics
        oCylinderStrip = oCylinderNode.AddTriangleStripGraphics

        ' Set the coordinates and index set for the cap.
        oCylinderStrip.CoordinateSet = oCoordSet
        oCylinderStrip.CoordinateIndexSet = oCylinderIndex

        Dim oAppearance As Asset
        oAppearance = oDoc.AppearanceAssets(1)

        oCylinderNode.Appearance = oAppearance

        ' update the display to see the results.
        ThisApplication.ActiveView.Update

        ' Define a normal for cap 1.
        Dim oCap1Normals As GraphicsNormalSet
        oCap1Normals = oDataSets.CreateNormalSet(1)
        Call oCap1Normals.Add(1, oTransGeom.CreateUnitVector(0, 0, -1))

        ' Assign the normals to the cap.
        oCap1TriangleFan.NormalSet = oCap1Normals

        ' Define a normal for cap 2.
        Dim oCap2Normals As GraphicsNormalSet
        oCap2Normals = oDataSets.CreateNormalSet(2)
        Call oCap2Normals.Add(1, oTransGeom.CreateUnitVector(0, 0, 1))

        ' Assign the normals to the cap.
        oCap2TriangleFan.NormalSet = oCap2Normals

        ' Create an array that contains the normals for each vertex of the cylinder.
        Dim adNormals(0 To ((iSideCount + 1) * 2) * 3 - 1) As Double
        dAngle = 0
        For i = 0 To iSideCount
            Dim dX As Double
            Dim dY As Double
            dX = Cos(dAngle)
            dY = Sin(dAngle)
            adNormals(i * 6 + 0) = dX
            adNormals(i * 6 + 1) = dY
            adNormals(i * 6 + 2) = 0
            adNormals(i * 6 + 3) = dX
            adNormals(i * 6 + 4) = dY
            adNormals(i * 6 + 5) = 0

            ' Increment the angle for the next normal.
            dAngle = dAngle + ((2 * 3.14159265358979) / iSideCount)
        Next

        ' Create and set the normal set for the cylinder.
        Dim oCylinderNormals As GraphicsNormalSet
        oCylinderNormals = oDataSets.CreateNormalSet(3)
        Call oCylinderNormals.PutNormals(adNormals)

        ' Assign the normals to the cylinder.
        oCylinderStrip.NormalSet = oCylinderNormals

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    End If
```

Anchored Client Grahics

Description

This sample demonstrates the creation of client graphics that is fully anchored in a view.

Code Samples

- VBA
- iLogic

This sample anchors some text graphics relative to the top-left corner of the view. Open a part or an assembly document and run the following sample.

Copy Code

```
Public Sub AnchoredClientGraphics()
  ' Set a reference to the document.
  Dim oDoc As Document
  Set oDoc = ThisApplication.ActiveDocument

  ' Set a reference to the component definition.
  ' This assumes that the active document is a part or an assembly.
  Dim oCompDef As ComponentDefinition
  Set oCompDef = oDoc.ComponentDefinition
```

```
  ' Attempt to get the existing client graphics object. If it exists
  ' delete it so the rest of the code can continue as if it never existed.
  Dim oClientGraphics As ClientGraphics
  On Error Resume Next
  Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Anchored Text")
  If Err.Number = 0 Then
    oClientGraphics.Delete
  End If
  On Error GoTo 0
  ThisApplication.ActiveView.Update

  ' Create a new ClientGraphics object.
  Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Anchored Text")

  ' Create a graphics node.
  Dim oNode As GraphicsNode
  Set oNode = oClientGraphics.AddNode(1)

  ' Create text graphics.
  Dim oTextGraphics As TextGraphics
  Set oTextGraphics = oNode.AddTextGraphics

  ' Set the properties of the text.
  oTextGraphics.Text = "Anchored text."
  oTextGraphics.Bold = True
  oTextGraphics.FontSize = 30
  Call oTextGraphics.PutTextColor(0, 255, 0)

  Dim oAnchorPoint As Point
  Set oAnchorPoint = ThisApplication.TransientGeometry.CreatePoint(1, 1, 1)

  ' Set the text's anchor in model space.
  oTextGraphics.Anchor = oAnchorPoint

  ' Anchor the text graphics in the view.
  Call oTextGraphics.SetViewSpaceAnchor( _
      oAnchorPoint, ThisApplication.TransientGeometry.CreatePoint2d(30, 30), _
      kTopLeftViewCorner)

  ' Update the view to see the text.
  ThisApplication.ActiveView.Update
End Sub
```

This sample anchors some text graphics relative to the top-left corner of the view. Open a part or an assembly document and run the following sample.

Copy Code

```
    ' Set a reference to the document.
  Dim oDoc As Document
  oDoc = ThisApplication.ActiveDocument

  ' Set a reference to the component definition.
  ' This assumes that the active document is a part or an assembly.
  Dim oCompDef As ComponentDefinition
  oCompDef = oDoc.ComponentDefinition

  ' Attempt to get the existing client graphics object. If it exists
  ' delete it so the rest of the code can continue as if it never existed.
  Dim oClientGraphics As ClientGraphics
  On Error Resume Next
  oClientGraphics = oCompDef.ClientGraphicsCollection.Item("Anchored Text")
  If Err.Number = 0 Then
    oClientGraphics.Delete
  End If
  On Error GoTo 0
  ThisApplication.ActiveView.Update

  ' Create a new ClientGraphics object.
  oClientGraphics = oCompDef.ClientGraphicsCollection.Add("Anchored Text")

  ' Create a graphics node.
  Dim oNode As GraphicsNode
  oNode = oClientGraphics.AddNode(1)

  ' Create text graphics.
  Dim oTextGraphics As TextGraphics
  oTextGraphics = oNode.AddTextGraphics

  ' Set the properties of the text.
  oTextGraphics.Text = "Anchored text."
  oTextGraphics.Bold = True
  oTextGraphics.FontSize = 30
  Call oTextGraphics.PutTextColor(0, 255, 0)

  Dim oAnchorPoint As Point
  oAnchorPoint = ThisApplication.TransientGeometry.CreatePoint(1, 1, 1)

  ' Set the text's anchor in model space.
  oTextGraphics.Anchor = oAnchorPoint

  ' Anchor the text graphics in the view.
  Call oTextGraphics.SetViewSpaceAnchor( _
      oAnchorPoint, ThisApplication.TransientGeometry.CreatePoint2d(30, 30), _
      kTopLeftViewCorner)

  ' Update the view to see the text.
  ThisApplication.ActiveView.Update
```

Client graphics - image in point graphics

Description

The following sample demonstrates creation of point client graphics with a custom image.

Code Samples

- VBA
- iLogic

The sample assumes that you've placed a MyImage.bmp in the C:\Temp\ directory.

Copy Code

```
Public Sub ImageInPointClientGraphics()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document.  If it doesn't create it.
    On Error Resume Next
    Dim oGraphicsData As GraphicsDataSets
    Set oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oGraphicsData.Delete
        oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID").Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        Set oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        Set oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Create an array that contains coordinates that define a set
        ' of outwardly spiraling points.
        Dim oPointCoords(2) As Double
        oPointCoords(0) = 1
        oPointCoords(1) = 1
        oPointCoords(2) = 0

        ' Assign the points into the coordinate set.
        Call oCoordSet.PutCoordinates(oPointCoords)

        ' Create an image set
        Dim oImageSet As GraphicsImageSet
        Set oImageSet = oDataSets.CreateImageSet(2)

        Dim oImage As IPictureDisp
        Set oImage = LoadPicture("C:\Temp\MyImage.bmp")

        Call oImageSet.Add(1, oImage)

        ' Create the ClientGraphics object.
        Dim oClientGraphics As ClientGraphics
        Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oPointNode As GraphicsNode
        Set oPointNode = oClientGraphics.AddNode(1)

        ' Create a PointGraphics object within the node.
        Dim oPointGraphics As PointGraphics
        Set oPointGraphics = oPointNode.AddPointGraphics

        ' Assign the coordinate set to the point graphics.
        oPointGraphics.CoordinateSet = oCoordSet

        ' Set a custom image
        Call oPointGraphics.SetCustomImage(oImageSet, 1)

        ' Update the view.
        ThisApplication.ActiveView.Update
    End If
End Sub
```

The sample assumes that you've placed a MyImage.bmp in the C:\Temp\ directory.

Copy Code

```
AddReference "stdole.dll"
Imports stdole
AddReference "System.Drawing.dll"
Sub Main
    Dim oDoc As Document
```

```
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document.  If it doesn't create it.
    On Error Resume Next
    Dim oGraphicsData As GraphicsDataSets
    oGraphicsData = oDoc.GraphicsDataSetsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0

        ' An existing client graphics object was successfully obtained so clean up.
        oGraphicsData.Delete
                oCompDef.ClientGraphicsCollection("SampleGraphicsID").Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        oDataSets = oDoc.GraphicsDataSetsCollection.Add("SampleGraphicsID")

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Create an array that contains coordinates that define a set
        ' of outwardly spiraling points.
        Dim oPointCoords(2) As Double
        oPointCoords(0) = 1
        oPointCoords(1) = 1
        oPointCoords(2) = 0

        ' Assign the points into the coordinate set.
        Call oCoordSet.PutCoordinates(oPointCoords)

        ' Create an image set
        Dim oImageSet As GraphicsImageSet
        oImageSet = oDataSets.CreateImageSet(2)

                Dim oImageFromFile As System.Drawing.Image
                oImageFromFile = System.Drawing.Image.FromFile("C:\Temp\MyImage.bmp")

                Dim oAxHostConverter As AxHostConverter = New AxHostConverter

        Dim oImage As stdole.IPictureDisp
        oImage = oAxHostConverter.GetIPictureDispFromImage(oImageFromFile)

        Call oImageSet.Add(1, oImage)

        ' Create the ClientGraphics object.
        Dim oClientGraphics As ClientGraphics
        oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oPointNode As GraphicsNode
        oPointNode = oClientGraphics.AddNode(1)

        ' Create a PointGraphics object within the node.
        Dim oPointGraphics As PointGraphics
        oPointGraphics = oPointNode.AddPointGraphics

        ' Assign the coordinate set to the point graphics.
        oPointGraphics.CoordinateSet = oCoordSet

        ' Set a custom image
        Call oPointGraphics.SetCustomImage(oImageSet, 1)

        ' Update the view.
        ThisApplication.ActiveView.Update
    End If
End Sub

Class AxHostConverter
        Inherits System.Windows.Forms.AxHost

Public Sub New()
        MyBase.New("{63109182-966B-4e3c-A8B2-8BC4A88D221C}")
End Sub

Public Function GetIPictureDispFromImage(ByVal image As System.Drawing.Image) As stdole.IPictureDisp
        Return MyBase.GetIPictureDispFromPicture(image)
End Function
End Class
```

Selection of Surface Graphics Primitives

Description

This demonstrates the ability to select client graphic primitives, by creating SurfaceGraphics and showing how you can select B-Rep entities within the graphics. You must have a part or assembly open and select a part of sat file which will be read in and displayed as client graphics. Depending on our responses to the program it will create the graphics so that only the node is selectable (which is all that was supported before), so that all of the primitives are selected, or so that only certain primitives are selectable (every other face in this case).

Code Samples

- VBA
- iLogic

To use this sample a part or assembly must be active.

Copy Code

```
Public Sub SelectablePrimitives()
    ' Make sure a part or assembly is active.
    If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Ther
        MsgBox "A part or assembly must be active."
        Exit Sub
    End If

    ' Get the document and it's associated definition.
    Dim doc As Document
    Set doc = ThisApplication.ActiveDocument
    Dim def As ComponentDefinition
    Set def = doc.ComponentDefinition

    ' Get an .ipt or SAT filename from the user.
    Dim dialog As FileDialog
    Dim filename As String
    Call ThisApplication.CreateFileDialog(dialog)
    With dialog
        .Filter = "Inventor Part File (*.ipt)|*.ipt|ACIS SAT File (*.sat)|*.sat"
        .FilterIndex = 0
        .ShowOpen
        If .filename = "" Then
            Exit Sub
        Else
            filename = .filename
        End If
    End With

    ' Get the transient B-Rep object.
    Dim tb As TransientBRep
    Set tb = ThisApplication.TransientBRep

    ' If a part was selected, open it inivisibly and get the body.
    Dim body As SurfaceBody
    If UCase(Right$(filename, 4)) = ".IPT" Then
        Dim newPart As PartDocument
        Set newPart = ThisApplication.Documents.Open(filename, False)

        ' Copy out the first body of the part.
        Set body = tb.Copy(newPart.ComponentDefinition.SurfaceBodies.Item(1))
    Else
        ' Read in the sat file.
        Dim bodies As SurfaceBodies
        Set bodies = tb.ReadFromFile(filename)

        Set body = bodies.Item(1)
    End If

    ' Create the client graphics object.
    Dim graphics As ClientGraphics
    On Error Resume Next
    Set graphics = def.ClientGraphicsCollection.Item("Test")
    If Err.Number = 0 Then
        graphics.Delete
    End If
    On Error GoTo 0

    Set graphics = def.ClientGraphicsCollection.Add("Test")
    Dim node As GraphicsNode
    Set node = graphics.AddNode(1)
    node.Selectable = True
    node.DisplayName = "My Solid"

    Dim answer As VbMsgBoxResult
    answer = MsgBox("Yes=Node Selectable, No=All Selectable, Cancel=Every other face selectable", vbYesNoCancel + vbQuestion)

    Dim solidGraphics As SurfaceGraphics
    Set solidGraphics = node.AddSurfaceGraphics(body)
    ThisApplication.ActiveView.Update

    If answer = vbYes Then
        solidGraphics.ChildrenAreSelectable = False
    Else
        solidGraphics.ChildrenAreSelectable = True

        If answer = vbCancel Then
            Dim i As Integer
            For i = 1 To solidGraphics.DisplayedVertices.Count
                solidGraphics.DisplayedVertices.Item(i).Selectable = False
            Next

            For i = 1 To solidGraphics.DisplayedEdges.Count
                solidGraphics.DisplayedEdges.Item(i).Selectable = False
            Next
```

```
            For i = 1 To solidGraphics.DisplayedFaces.Count
                If i Mod 2 = 0 Then
                    solidGraphics.DisplayedFaces.Item(i).Selectable = False
                End If
            Next
        End If
    End If
End Sub
```

To use this sample a part or assembly must be active.

Copy Code

```
' Make sure a part or assembly is active.
If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Thei
    MsgBox("A part or assembly must be active.")
    Exit Sub
End If

' Get the document and it's associated definition.
Dim doc As Document
doc = ThisApplication.ActiveDocument
Dim def As ComponentDefinition
def = doc.ComponentDefinition

' Get an .ipt or SAT filename from the user.
Dim dialog As FileDialog
Dim filename As String
Call ThisApplication.CreateFileDialog(dialog)
With dialog
    .Filter = "Inventor Part File (*.ipt)|*.ipt|ACIS SAT File (*.sat)|*.sat"
    .FilterIndex = 0
    .ShowOpen
    If .filename = "" Then
        Exit Sub
    Else
        filename = .filename
    End If
End With

' Get the transient B-Rep object.
Dim tb As TransientBRep
tb = ThisApplication.TransientBRep

' If a part was selected, open it inivisibly and get the body.
Dim body As SurfaceBody
If UCase(Right$(filename, 4)) = ".IPT" Then
    Dim newPart As PartDocument
    newPart = ThisApplication.Documents.Open(filename, False)

    ' Copy out the first body of the part.
    body = tb.Copy(newPart.ComponentDefinition.SurfaceBodies.Item(1))
Else
    ' Read in the sat file.
    Dim bodies As SurfaceBodies
    bodies = tb.ReadFromFile(filename)

    body = bodies.Item(1)
End If

' Create the client graphics object.
Dim graphics As ClientGraphics
On Error Resume Next
graphics = def.ClientGraphicsCollection.Item("Test")
If Err.Number = 0 Then
    graphics.Delete
End If
On Error GoTo 0

graphics = def.ClientGraphicsCollection.Add("Test")
Dim node As GraphicsNode
node = graphics.AddNode(1)
node.Selectable = True
node.DisplayName = "My Solid"

Dim answer As MsgBoxResult
answer = MsgBox("Yes=Node Selectable, No=All Selectable, Cancel=Every other face selectable", vbYesNoCancel + vbQuestion)

Dim solidGraphics As SurfaceGraphics
solidGraphics = node.AddSurfaceGraphics(body)
ThisApplication.ActiveView.Update

If answer = vbYes Then
    solidGraphics.ChildrenAreSelectable = False
Else
    solidGraphics.ChildrenAreSelectable = True

    If answer = vbCancel Then
        Dim i As Integer
        For i = 1 To solidGraphics.DisplayedVertices.Count
            solidGraphics.DisplayedVertices.Item(i).Selectable = False
        Next

        For i = 1 To solidGraphics.DisplayedEdges.Count
            solidGraphics.DisplayedEdges.Item(i).Selectable = False
        Next

        For i = 1 To solidGraphics.DisplayedFaces.Count
            If i Mod 2 = 0 Then
                solidGraphics.DisplayedFaces.Item(i).Selectable = False
            End If
        Next
    End If
End If
```

Create curve primitives

Description

This sample demonstrates the creation of curve primitives (lines, arcs, circles, etc.) using client graphics.

Code Samples

- VBA
- iLogic

To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics.

Copy Code

```
Public Sub ClientGraphicsPrimitives()
    Dim odoc As Document
    Set odoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document. If it doesn't create it.
    On Error Resume Next
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oClientGraphics.Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create the ClientGraphics object.
        Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oCurvesNode As GraphicsNode
        Set oCurvesNode = oClientGraphics.AddNode(1)

        Dim oCenter As Point
        Set oCenter = oTransGeom.CreatePoint(1, 1, 0)

        Dim oNormal As UnitVector
        Set oNormal = oTransGeom.CreateUnitVector(0, 0, 1)

        ' Create a transient circle object
        Dim oCircle As Inventor.Circle
        Set oCircle = oTransGeom.CreateCircle(oCenter, oNormal, 1)

        ' Create a circle graphics object within the node.
        Dim oCircleGraphics As CurveGraphics
        Set oCircleGraphics = oCurvesNode.AddCurveGraphics(oCircle)

        Dim oReference As UnitVector
        Set oReference = oTransGeom.CreateUnitVector(-1, 0, 0)

        ' Create a transient arc object
        Dim oArc1 As Arc3d
        Set oArc1 = oTransGeom.CreateArc3d(oCenter, oNormal, oReference, 3, 3.14159 / 4, 3.14159 / 2)

        ' Create an arc graphics object within the node.
        Dim oArcGraphics1 As CurveGraphics
        Set oArcGraphics1 = oCurvesNode.AddCurveGraphics(oArc1)

        ' Create a transient arc object
        Dim oArc2 As Arc3d
        Set oArc2 = oTransGeom.CreateArc3d(oCenter, oNormal, oReference, 3, 3.14159 + (3.14159 / 4), 3.14159 / 2)

        ' Create an arc graphics object within the node.
        Dim oArcGraphics2 As CurveGraphics
        Set oArcGraphics2 = oCurvesNode.AddCurveGraphics(oArc2)

        ' Create a transient line segment object
        Dim oLineSegment1 As LineSegment
        Set oLineSegment1 = oTransGeom.CreateLineSegment(oArc1.StartPoint, oArc2.EndPoint)

        ' Create an line graphics object within the node.
        Dim oLineGraphics1 As CurveGraphics
        Set oLineGraphics1 = oCurvesNode.AddCurveGraphics(oLineSegment1)

        ' Create a transient line segment object
        Dim oLineSegment2 As LineSegment
        Set oLineSegment2 = oTransGeom.CreateLineSegment(oArc2.StartPoint, oArc1.EndPoint)
```

```
        ' Create an line graphics object within the node.
        Dim oLineGraphics2 As CurveGraphics
        Set oLineGraphics2 = oCurvesNode.AddCurveGraphics(oLineSegment2)

        ' Update the view to see the resulting curves.
        ThisApplication.ActiveView.Update
    End If
End Sub
```

To use the sample you need to have an assembly or part document open. The program has two behaviors: the first time it is run it will draw the graphics. The second time it is run it deletes the previously drawn graphics.

Copy Code

```
    Dim odoc As Document
    odoc = ThisApplication.ActiveDocument

    ' Set a reference to component definition of the active document.
    ' This assumes that a part or assembly document is active.
    Dim oCompDef As ComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Check to see if the test graphics data object already exists.
    ' If it does clean up by removing all associated of the client graphics
    ' from the document. If it doesn't create it.
    On Error Resume Next
    Dim oClientGraphics As ClientGraphics
    oClientGraphics = oCompDef.ClientGraphicsCollection.Item("SampleGraphicsID")
    If Err.Number = 0 Then
        On Error GoTo 0
        ' An existing client graphics object was successfully obtained so clean up.
        oClientGraphics.Delete

        ' update the display to see the results.
        ThisApplication.ActiveView.Update
    Else
        Err.Clear
        On Error GoTo 0

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        oTransGeom = ThisApplication.TransientGeometry

        ' Create the ClientGraphics object.
        oClientGraphics = oCompDef.ClientGraphicsCollection.Add("SampleGraphicsID")

        ' Create a new graphics node within the client graphics objects.
        Dim oCurvesNode As GraphicsNode
        oCurvesNode = oClientGraphics.AddNode(1)

        Dim oCenter As Point
        oCenter = oTransGeom.CreatePoint(1, 1, 0)

        Dim oNormal As UnitVector
        oNormal = oTransGeom.CreateUnitVector(0, 0, 1)

        ' Create a transient circle object
        Dim oCircle As Inventor.Circle
        oCircle = oTransGeom.CreateCircle(oCenter, oNormal, 1)

        ' Create a circle graphics object within the node.
        Dim oCircleGraphics As CurveGraphics
        oCircleGraphics = oCurvesNode.AddCurveGraphics(oCircle)

        Dim oReference As UnitVector
        oReference = oTransGeom.CreateUnitVector(-1, 0, 0)

        ' Create a transient arc object
        Dim oArc1 As Arc3d
        oArc1 = oTransGeom.CreateArc3d(oCenter, oNormal, oReference, 3, 3.14159 / 4, 3.14159 / 2)

        ' Create an arc graphics object within the node.
        Dim oArcGraphics1 As CurveGraphics
        oArcGraphics1 = oCurvesNode.AddCurveGraphics(oArc1)

        ' Create a transient arc object
        Dim oArc2 As Arc3d
        oArc2 = oTransGeom.CreateArc3d(oCenter, oNormal, oReference, 3, 3.14159 + (3.14159 / 4), 3.14159 / 2)

        ' Create an arc graphics object within the node.
        Dim oArcGraphics2 As CurveGraphics
        oArcGraphics2 = oCurvesNode.AddCurveGraphics(oArc2)

        ' Create a transient line segment object
        Dim oLineSegment1 As LineSegment
        oLineSegment1 = oTransGeom.CreateLineSegment(oArc1.StartPoint, oArc2.EndPoint)

        ' Create an line graphics object within the node.
        Dim oLineGraphics1 As CurveGraphics
        oLineGraphics1 = oCurvesNode.AddCurveGraphics(oLineSegment1)

        ' Create a transient line segment object
        Dim oLineSegment2 As LineSegment
        oLineSegment2 = oTransGeom.CreateLineSegment(oArc2.StartPoint, oArc1.EndPoint)

        ' Create an line graphics object within the node.
        Dim oLineGraphics2 As CurveGraphics
        oLineGraphics2 = oCurvesNode.AddCurveGraphics(oLineSegment2)

        ' Update the view to see the resulting curves.
        ThisApplication.ActiveView.Update
    End If
```

Update iProperty values using Apprentice

Description

Updates some iProperty values using Apprentice. The document specified in the code for the Open method must exist.

Code Samples

- VB.Net

Create a new VB.Net project and add a reference to the Inventor interop. Copy the code below into the project and call the function. For a Forms application you can call this when a button is clicked, or from a console application you can call it from the Main function. Please note that the sample works with Inventor version since Inventor 2022.2 or later versions which have the ApprenticeServerDocument.FilePropertySets, to work with Inventor 2022.1 or previous version you should change the ApprenticeServerDocument.FilePropertySets to ApprenticeServerDocument.PropertySets.

Copy Code

```
Public Sub ApprenticeUpdate()
    ' Declare a variable for Apprentice.
    Dim invApprentice As New Inventor.ApprenticeServerComponent

    ' Open a document using Apprentice.
    Dim invDoc As Inventor.ApprenticeServerDocument
    invDoc = invApprentice.Open("C:\Temp\Part1.ipt")

    ' Get the design tracking property set.
    Dim invDTProperties As Inventor.PropertySet
    invDTProperties = invDoc.FilePropertySets.Item("Design Tracking Properties")

    ' Edit the values of a couple of properties.
    invDTProperties.Item("Checked By").Value = "Bob"
    invDTProperties.Item("Date Checked").Value = Now

    ' Save the changes.
    invDoc.FilePropertySets.FlushToFile()

    ' Close the document and release all references.
    invDoc = Nothing
    invApprentice.Close()
    invApprentice = Nothing
End Sub
```

Create custom iProperties

Description

Creates custom iProperties of various types. A document must be open when this sample is run.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateCustomProperties()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the user defined (custom) property set.
    Dim invCustomPropertySet As PropertySet
    Set invCustomPropertySet = invDoc.PropertySets.Item("Inventor User Defined Properties")

    ' Declare some variables that will contain the various values.
    Dim strText As String
    Dim dblValue As Double
    Dim dtDate As Date
    Dim blYesOrNo As Boolean

    ' Set values for the variables.
    strText = "Some sample text."
    dblValue = 3.14159
    dtDate = Now
    blYesOrNo = True

    ' Create the properties.
    Dim invProperty As Property
    Set invProperty = invCustomPropertySet.Add(strText, "Test Test")
    Set invProperty = invCustomPropertySet.Add(dblValue, "Test Value")
    Set invProperty = invCustomPropertySet.Add(dtDate, "Test Date")
    Set invProperty = invCustomPropertySet.Add(blYesOrNo, "Test Yes or No")
End Sub
```

Copy Code

```
    ' Get the active document.
    Dim invDoc As Document
    invDoc = ThisApplication.ActiveDocument

    ' Get the user defined (custom) property set.
```

```
    Dim invCustomPropertySet As PropertySet
    invCustomPropertySet = invDoc.PropertySets.Item("Inventor User Defined Properties")

    ' Declare some variables that will contain the various values.
    Dim strText As String
    Dim dblValue As Double
    Dim dtDate As Date
    Dim blYesOrNo As Boolean

    ' Set values for the variables.
    strText = "Some sample text."
    dblValue = 3.14159
    dtDate = Now
    blYesOrNo = True

    ' Create the properties.
    Dim invProperty As Inventor.Property
    invProperty = invCustomPropertySet.Add(strText, "Test Test")
    invProperty = invCustomPropertySet.Add(dblValue, "Test Value")
    invProperty = invCustomPropertySet.Add(dtDate, "Test Date")
    invProperty = invCustomPropertySet.Add(blYesOrNo, "Test Yes or No")
```

Create or update custom iProperty

Description

This example creates a custom iProperty if it doesn't exist and updates the value if it does already exist. A part document must be open before runnin the sample.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub UpdateVolume()
    ' Get the active part document.
    Dim invPartDoc As PartDocument
    Set invPartDoc = ThisApplication.ActiveDocument

    ' Get the volume of the part. This will be returned in
    ' cubic centimeters.
    Dim dVolume As Double
    dVolume = invPartDoc.ComponentDefinition.MassProperties.Volume

    ' Get the UnitsOfMeasure object which is used to do unit conversions.
    Dim oUOM As UnitsOfMeasure
    Set oUOM = invPartDoc.UnitsOfMeasure

    ' Convert the volume to the current document units.
    Dim strVolume As String
    strVolume = oUOM.GetStringFromValue(dVolume, oUOM.GetStringFromType(oUOM.LengthUnits) & "^3")

    ' Get the custom property set.
    Dim invCustomPropertySet As PropertySet
    Set invCustomPropertySet = invPartDoc.PropertySets.Item("Inventor User Defined Properties")

    ' Attempt to get an existing custom property named "Volume".
    On Error Resume Next
    Dim invVolumeProperty As Property
    Set invVolumeProperty = invCustomPropertySet.Item("Volume")
    If Err.Number <> 0 Then
        ' Failed to get the property, which means it doesn't exist
        ' so we'll create it.
        Call invCustomPropertySet.Add(strVolume, "Volume")
    Else
        ' Got the property so update the value.
        invVolumeProperty.value = strVolume
    End If
End Sub
```

Copy Code

```
    ' Get the active part document.
    Dim invPartDoc As PartDocument
    invPartDoc = ThisApplication.ActiveDocument

    ' Get the volume of the part. This will be returned in
    ' cubic centimeters.
    Dim dVolume As Double
    dVolume = invPartDoc.ComponentDefinition.MassProperties.Volume

    ' Get the UnitsOfMeasure object which is used to do unit conversions.
    Dim oUOM As UnitsOfMeasure
    oUOM = invPartDoc.UnitsOfMeasure

    ' Convert the volume to the current document units.
    Dim strVolume As String
    strVolume = oUOM.GetStringFromValue(dVolume, oUOM.GetStringFromType(oUOM.LengthUnits) & "^3")

    ' Get the custom property set.
    Dim invCustomPropertySet As PropertySet
    invCustomPropertySet = invPartDoc.PropertySets.Item("Inventor User Defined Properties")

    ' Attempt to get an existing custom property named "Volume".
    On Error Resume Next
    Dim invVolumeProperty As Inventor.Property
```

```
    invVolumeProperty = invCustomPropertySet.Item("Volume")
    If Err.Number <> 0 Then
        ' Failed to get the property, which means it doesn't exist
        ' so we'll create it.
        Call invCustomPropertySet.Add(strVolume, "Volume")
    Else
        ' Got the property so update the value.
        invVolumeProperty.value = strVolume
    End If
```

Get value of iProperty

Description

Demonstrates getting the values of the "Part Number" iProperty. Any property can be retrieved by accesing the correct property set and property.

A document must be open when this sample is run.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub GetPropertySample()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the design tracking property set.
    Dim invDesignInfo As PropertySet
    Set invDesignInfo = invDoc.PropertySets.Item("Design Tracking Properties")

    ' Get the part number property.
    Dim invPartNumberProperty As Property
    Set invPartNumberProperty = invDesignInfo.Item("Part Number")

    MsgBox "Part Number: " & invPartNumberProperty.value
End Sub
```

Copy Code

```
    ' Get the active document.
    Dim invDoc As Document
    invDoc = ThisApplication.ActiveDocument

    ' Get the design tracking property set.
    Dim invDesignInfo As PropertySet
    invDesignInfo = invDoc.PropertySets.Item("Design Tracking Properties")

    ' Get the part number property.
    Dim invPartNumberProperty As Inventor.Property
    invPartNumberProperty = invDesignInfo.Item("Part Number")

    MsgBox("Part Number: " & invPartNumberProperty.value)
```

Printing - Drawing Print

Description

This sample demonstrates how to print a drawing, setting specifics such as sheet range.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PrintDrawing()
    ' Set a reference to the print manager object of the active document.
    ' This will fail if a drawing document is not active.
    Dim oPrintMgr As DrawingPrintManager
    Set oPrintMgr = ThisApplication.ActiveDocument.PrintManager

    ' Get the name of the printer that will be used.
    If MsgBox("Using printer """ & oPrintMgr.Printer & """  Do you want to continue?", vbYesNo + vbQuestion) = vbNo Then
        ' Change to another printer.
        Dim sPrinterName As String
        sPrinterName = InputBox("Enter name of new printer:", "New Printer")
        If sPrinterName = "" Then
            Exit Sub
        Else
            oPrintMgr.Printer = sPrinterName
        End If
    End If
```

```
    ' Set to print in color.
    oPrintMgr.ColorMode = kPrintColorPalette

    ' Set to print two copies.
    oPrintMgr.NumberOfCopies = 2

    ' Set to print using portrait orientation.
    oPrintMgr.Orientation = kPortraitOrientation

    ' Set the paper size.
    oPrintMgr.PaperSize = kPaperSize11x17

    ' Set to print all sheets.
    oPrintMgr.PrintRange = kPrintAllSheets

    ' Set to print full scale.
    oPrintMgr.ScaleMode = kPrintFullScale

    ' Submit the print.
    oPrintMgr.SubmitPrint

    ' Change the number of copies to 1.
    oPrintMgr.NumberOfCopies = 1

    ' Change the paper size to a custom size.  The units are in centimeters.
    oPrintMgr.PaperSize = kPaperSizeCustom
    oPrintMgr.PaperHeight = 15
    oPrintMgr.PaperWidth = 10

    ' Get and set the current sheet range.
    Dim iFromSheet As Long
    Dim iToSheet As Long
    Call oPrintMgr.GetSheetRange(iFromSheet, iToSheet)

    MsgBox "Current sheet range is " & iFromSheet & " to " & iToSheet & Chr(13) & _
           "Setting to print sheets 1-2."

    ' Change the print range to print sheets 1 through 2.
    oPrintMgr.PrintRange = kPrintSheetRange
    Call oPrintMgr.SetSheetRange(1, 2)

    ' Submit the print.
    oPrintMgr.SubmitPrint
End Sub
```

Copy Code

```
    ' Set a reference to the print manager object of the active document.
    ' This will fail if a drawing document is not active.
    Dim oPrintMgr As DrawingPrintManager
    oPrintMgr = ThisApplication.ActiveDocument.PrintManager

    ' Get the name of the printer that will be used.
    If MsgBox("Using printer """ & oPrintMgr.Printer & """  Do you want to continue?", vbYesNo + vbQuestion) = vbNo Then
        ' Change to another printer.
        Dim sPrinterName As String
        sPrinterName = InputBox("Enter name of new printer:", "New Printer")
        If sPrinterName = "" Then
            Exit Sub
        Else
            oPrintMgr.Printer = sPrinterName
        End If
    End If

    ' Set to print in color.
    oPrintMgr.ColorMode = kPrintColorPalette

    ' Set to print two copies.
    oPrintMgr.NumberOfCopies = 2

    ' Set to print using portrait orientation.
    oPrintMgr.Orientation = kPortraitOrientation

    ' Set the paper size.
    oPrintMgr.PaperSize = kPaperSize11x17

    ' Set to print all sheets.
    oPrintMgr.PrintRange = kPrintAllSheets

    ' Set to print full scale.
    oPrintMgr.ScaleMode = kPrintFullScale

    ' Submit the print.
    oPrintMgr.SubmitPrint

    ' Change the number of copies to 1.
    oPrintMgr.NumberOfCopies = 1

    ' Change the paper size to a custom size.  The units are in centimeters.
    oPrintMgr.PaperSize = kPaperSizeCustom
    oPrintMgr.PaperHeight = 15
    oPrintMgr.PaperWidth = 10

    ' Get and set the current sheet range.
    Dim iFromSheet As Long
    Dim iToSheet As Long
    Call oPrintMgr.GetSheetRange(iFromSheet, iToSheet)

    MsgBox("Current sheet range is " & iFromSheet & " to " & iToSheet & Chr(13) & _
           "Setting to print sheets 1-2.")

    ' Change the print range to print sheets 1 through 2.
    oPrintMgr.PrintRange = kPrintSheetRange
    Call oPrintMgr.SetSheetRange(1, 2)
```

```
    ' Submit the print.
    oPrintMgr.SubmitPrint
```

Drive the camera

Description

This sample will fly the camera around the model. To simplify the code, the target is hard coded at the origin and the up vector is the positive Z.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub RotateCamera()
    ' Get the active camera.
    Dim cam As Camera
    Set cam = ThisApplication.ActiveView.Camera

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Define the number of steps in the animation.
    Dim steps As Integer
    steps = 360

    ' Define the distance between the eye and target.
    Dim eyeDistance As Double
    eyeDistance = 15

    ' Calculate pi.
    Dim pi As Double
    pi = Atn(1) * 4

    ' Iterate the specified number of steps.
    Dim i As Integer
    For i = 1 To steps
        ' Calculate the x and y coordinates of the eye.
        Dim x As Double
        Dim y As Double
        x = eyeDistance * Cos(i / steps * (2 * pi))
        y = eyeDistance * Sin(i / steps * (2 * pi))

        ' Set the eye with a hard coded z value.
        cam.Eye = tg.CreatePoint(x, y, 3)

        ' Define the up vector as positive z.
        cam.UpVector = tg.CreateUnitVector(0, 0, 1)

        ' Apply the current camera definition to the view.
        cam.ApplyWithoutTransition
    Next
End Sub
```

Copy Code

```
    ' Get the active camera.
    Dim cam As Camera
    cam = ThisApplication.ActiveView.Camera

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Define the number of steps in the animation.
    Dim steps As Integer
    steps = 360

    ' Define the distance between the eye and target.
    Dim eyeDistance As Double
    eyeDistance = 15

    ' Calculate pi.
    Dim pi As Double
    pi = Math.Atan(1) * 4

    ' Iterate the specified number of steps.
    Dim i As Integer
    For i = 1 To steps
        ' Calculate the x and y coordinates of the eye.
        Dim x As Double
        Dim y As Double
        x = eyeDistance * Cos(i / steps * (2 * pi))
        y = eyeDistance * Sin(i / steps * (2 * pi))

        ' Set the eye with a hard coded z value.
        cam.Eye = tg.CreatePoint(x, y, 3)

        ' Define the up vector as positive z.
        cam.UpVector = tg.CreateUnitVector(0, 0, 1)

        ' Apply the current camera definition to the view.
        cam.ApplyWithoutTransition
    Next
```

Pack and go

Description

This sample demonstrates using pack and go interop to create a package.

Code Samples

- VB.Net
- C#
- C++

Create a VB.Net project and add the reference to the pack and go type interop binary Autodesk.PackAndGo.Interop.dll which is in the Bin folder of Inventor installation folder.

Copy Code

```
Public Sub PackAndGoSample()
    Dim oPacknGoComp As New PackAndGoLib.PackAndGoComponent

    Dim oPacknGo As PackAndGoLib.PackAndGo
    oPacknGo = oPacknGoComp.CreatePackAndGo("C:\Temp\Source\Assembly1.iam", "C:\Temp\Destination")

    ' Set the design project. This defaults to the current active project.
    oPacknGo.ProjectFile = "C:\Temp\Source\Test.ipj"

    Dim sRefFiles = New String() {}
    Dim sMissFiles = New Object

    ' Set the options
    oPacknGo.IsSkippingLibraries = True
    oPacknGo.IsSkippingStyles = True
    oPacknGo.IsSkippingTemplates = True
    oPacknGo.IsCollectingWorkgroups = False
    oPacknGo.IsKeepingFolderHierarchy = True
    oPacknGo.IncludeLinkedFiles = True

    ' Get the referenced files
    oPacknGo.SearchForReferencedFiles(sRefFiles, sMissFiles)

    ' Add the referenced files for package
    oPacknGo.AddFilesToPackage (sRefFiles)

    ' Start the pack and go to create the package
    oPacknGo.CreatePackage()
End Sub
```

Create a C#.Net project and add the reference to the pack and go type interop binary Autodesk.PackAndGo.Interop.dll which is in the Bin folder of Inventor installation folder.

Copy Code

```
public void PackAndGoSample()
{
    PackAndGoComponent packAndGoComp = new PackAndGoComponent();
    PackAndGo packAndGo=packAndGoComp.CreatePackAndGo("C:\\Temp\\Source\\Assembly1.iam","C:\\Temp\\Destination");

    // Set the design project. This defaults to the current active project.
    packAndGo.ProjectFile = "C:\\Temp\\Source\\Test.ipj";

    string[] refFiles = new string[]{};
    object refMissFiles = new object();

    // Set the options
    packAndGo.IsSkippingLibraries = true;
    packAndGo.IsSkippingStyles = true;
    packAndGo.IsSkippingTemplates = true;
    packAndGo.IsCollectingWorkgroups = false;
    packAndGo.IsKeepingFolderHierarchy = true;
    packAndGo.IncludeLinkedFiles = true;

    // Get the referenced files
    packAndGo.SearchForReferencedFiles(out refFiles,out refMissFiles);

    // Add the referenced files for package
    packAndGo.AddFilesToPackage(refFiles);

    // Start the pack and go to create the package
    packAndGo.CreatePackage();
}
```

Create a C++ project and add the include the Bin folder of Inventor installation folder where the DTPackAndGo.tlb is located.

Copy Code

```
#import "DTPackAndGo.tlb" no_namespace named_guids \
     raw_dispinterfaces raw_method_prefix("") \
     high_method_prefix("Method")

void PackAndGoSample()
{
  HRESULT Result = NOERROR;

  CComPtr<IPackAndGoComponent> packAndGoComp;
  Result = packAndGoComp.CoCreateInstance(CLSID_PackAndGoComponent);

  CComPtr<IPackAndGo> packAndGo =
    packAndGoComp->MethodCreatePackAndGo(
```

```
      L"C:\\Temp\\Source\\Assembly1.iam", L"C:\\Temp\\Destination");

  // Set the design project. This defaults to the
  // current active project.
  packAndGo->ProjectFile = L"C:\\Temp\\Source\\Test.ipj";

  // Set the options
  packAndGo->SkipLibraries = true;
  packAndGo->SkipStyles = true;
  packAndGo->SkipTemplates = true;
  packAndGo->CollectWorkgroups = false;
  packAndGo->KeepFolderHierarchy = true;
  packAndGo->IncludeLinkedFiles = true;

  SAFEARRAY * refFiles = NULL;
  VARIANT refMissFiles;

  // Get the referenced files
  packAndGo->MethodSearchForReferencedFiles(
    &refFiles, &refMissFiles);

  // Add the referenced files for package
  packAndGo->MethodAddFilesToPackage(&refFiles);

  // Start the pack and go to create the package
  // OverrideExistingFiles = TRUE
  packAndGo->MethodCreatePackage(VARIANT_TRUE);
}
```

Printing - Part or Assembly

Description

This sample demonstrates how to print a Part or Assembly document.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PrintPartOrAssembly()
    ' Set a reference to the print manager object of the active document.
    Dim oPrintMgr As PrintManager
    Set oPrintMgr = ThisApplication.ActiveDocument.PrintManager

    ' Get the name of the printer that will be used.
    If MsgBox("Using printer """ & oPrintMgr.Printer & """  Do you want to continue?", vbYesNo + vbQuestion) = vbNo Then
        ' Change to another printer.
        Dim sPrinterName As String
        sPrinterName = InputBox("Enter name of new printer:", "New Printer")
        If sPrinterName = "" Then
            Exit Sub
        Else
            oPrintMgr.Printer = sPrinterName
        End If
    End If

    ' Set to print in color.
    oPrintMgr.ColorMode = kPrintColorPalette

    ' Set to print two copies.
    oPrintMgr.NumberOfCopies = 2

    ' Set to print using portrait orientation.
    oPrintMgr.Orientation = kPortraitOrientation

    ' Set the paper size.
    oPrintMgr.PaperSize = kPaperSize11x17

    ' Submit the print.
    oPrintMgr.SubmitPrint

    ' Change the number of copies to 1.
    oPrintMgr.NumberOfCopies = 1

    ' Change the paper size to a custom size.  The units are in centimeters.
    oPrintMgr.PaperSize = kPaperSizeCustom
    oPrintMgr.PaperHeight = 15
    oPrintMgr.PaperWidth = 10

    ' Submit the print.
    oPrintMgr.SubmitPrint
End Sub
```

Copy Code

```
    ' Set a reference to the print manager object of the active document.
    Dim oPrintMgr As PrintManager
    oPrintMgr = ThisApplication.ActiveDocument.PrintManager

    ' Get the name of the printer that will be used.
    If MsgBox("Using printer """ & oPrintMgr.Printer & """  Do you want to continue?", vbYesNo + vbQuestion) = vbNo Then
        ' Change to another printer.
        Dim sPrinterName As String
        sPrinterName = InputBox("Enter name of new printer:", "New Printer")
```

```
        If sPrinterName = "" Then
            Exit Sub
        Else
            oPrintMgr.Printer = sPrinterName
        End If
    End If

    ' Set to print in color.
    oPrintMgr.ColorMode = kPrintColorPalette

    ' Set to print two copies.
    oPrintMgr.NumberOfCopies = 2

    ' Set to print using portrait orientation.
    oPrintMgr.Orientation = kPortraitOrientation

    ' Set the paper size.
    oPrintMgr.PaperSize = kPaperSize11x17

    ' Submit the print.
    oPrintMgr.SubmitPrint

    ' Change the number of copies to 1.
    oPrintMgr.NumberOfCopies = 1

    ' Change the paper size to a custom size.  The units are in centimeters.
    oPrintMgr.PaperSize = kPaperSizeCustom
    oPrintMgr.PaperHeight = 15
    oPrintMgr.PaperWidth = 10

    ' Submit the print.
    oPrintMgr.SubmitPrint
```

Browser Search Box Sample

Description

This sample demonstrates how to use the search box in a document's browser pane.

Code Samples

- VBA
- iLogic

Open a document in Inventor first before running below sample.

Copy Code

```
Sub SearchBoxSample()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    Dim oPane As BrowserPane
    ' Get the BrowserPane that support the search box.
    If oDoc.DocumentType = kPartDocumentObject Then
        Set oPane = oDoc.BrowserPanes("PmDefault")
    ElseIf oDoc.DocumentType = kAssemblyDocumentObject Then
        Set oPane = oDoc.BrowserPanes("AmBrowserArrangement")
    ElseIf oDoc.DocumentType = kDrawingDocumentObject Then
        Set oPane = oDoc.BrowserPanes("DlHierarchy")
    ElseIf oDoc.DocumentType = kPresentationDocumentObject Then
        Set oPane = oDoc.BrowserPanes("DxHierarchy")
    End If

    Dim oSearchBox As SearchBox
    Set oSearchBox = oPane.SearchBox

    ' Enable the search box and display it in the broser pane for search text.
    oSearchBox.Enabled = True
    oSearchBox.Visible = True
    Call oSearchBox.Search("Part")
    oPane.Activate
End Sub
```

Open a document in Inventor first before running below sample.

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    Dim oPane As BrowserPane
    ' Get the BrowserPane that support the search box.
    If oDoc.DocumentType = kPartDocumentObject Then
        oPane = oDoc.BrowserPanes("PmDefault")
    ElseIf oDoc.DocumentType = kAssemblyDocumentObject Then
        oPane = oDoc.BrowserPanes("AmBrowserArrangement")
    ElseIf oDoc.DocumentType = kDrawingDocumentObject Then
        oPane = oDoc.BrowserPanes("DlHierarchy")
    ElseIf oDoc.DocumentType = kPresentationDocumentObject Then
        oPane = oDoc.BrowserPanes("DxHierarchy")
    End If

    Dim oSearchBox As SearchBox
    oSearchBox = oPane.SearchBox

    ' Enable the search box and display it in the broser pane for search text.
    oSearchBox.Enabled = True
    oSearchBox.Visible = True
```

```
    Call oSearchBox.Search("Part")
    oPane.Activate
```

Show help sample with local help

Description

The sample demonstrate how to set the help context/topic. Before running the sample you have to change the helpFile to specify it a correct Inventor API documentation path.

Code Samples

- VBA
- iLogic

The VBA sample demonstrate how to set the help context/topic to display help from a local chm file. Before running the sample you have to change the helpFile to specify it a with correct Inventor API documentation path.

Copy Code

```
Public Sub ShowHelpSampleWithLocalHelp()
    Dim helpFile As String
    helpFile = "C:\Users\Public\Documents\Autodesk\Inventor 20xx\Local Help\admapi_xx_0.chm"
    Call ThisApplication.HelpManager.DisplayHelpTopic(helpFile, "HTML\AliasFreeformFeature_ConsumeInputs.htm")
    Call ThisApplication.HelpManager.DisplayHelpContext(helpFile, 83886873)
End Sub
```

The VBA sample demonstrate how to set the help context/topic to display help from a local chm file. Before running the sample you have to change the helpFile to specify it a with correct Inventor API documentation path.

Copy Code

```
    Dim helpFile As String
    helpFile = "C:\Users\Public\Documents\Autodesk\Inventor 20xx\Local Help\admapi_xx_0.chm"
    Call ThisApplication.HelpManager.DisplayHelpTopic(helpFile, "HTML\AliasFreeformFeature_ConsumeInputs.htm")
    Call ThisApplication.HelpManager.DisplayHelpContext(helpFile, 83886873)
```

Show online help sample for file dialog

Description

The sample demonstrate how to set the online help for a file dialog.

Code Samples

- VBA
- iLogic

The VBA sample demonstrates how to set the online help page for a file dialog.

Copy Code

```
Public Sub ShowOnlineHelpForFileDialog()
    Dim helpWebsite As String
    helpWebsite = "www.google.com"

    Dim oFileDialog As FileDialog
    ThisApplication.CreateFileDialog oFileDialog

    ' when set the help context to an online website, the context should be specified as -1
    oFileDialog.SetHelpContext helpWebsite, -1

    oFileDialog.ShowOpen
End Sub
```

The VBA sample demonstrates how to set the online help page for a file dialog.

Copy Code

```
    Dim helpWebsite As String
    helpWebsite = "www.google.com"

    Dim oFileDialog As FileDialog
    ThisApplication.CreateFileDialog(oFileDialog)

    ' when set the help context to an online website, the context should be specified as -1
    oFileDialog.SetHelpContext(helpWebsite, -1)

    oFileDialog.ShowOpen
```

Inventor theme change sample

Description

This sample demonstrates how to change the Inventor theme.

Code Samples

- VBA
- iLogic

This sample demonstrates how to change the Inventor theme.

Copy Code

```
Sub SwitchThemeSample()
    Dim oThemeManager As ThemeManager
    Set oThemeManager = ThisApplication.ThemeManager

    Dim oTheme As Theme, i As Long: i = 1
    Dim sNames As String, sName As String
    For Each oTheme In oThemeManager.Themes
        sNames = sNames & i & "): " & oTheme.Name & vbCrLf
        i = i + 1
    Next

    sName = InputBox("Which theme would you like to set: " & vbCrLf & vbCrLf & sNames, "Inventor")
    i = CInt(sName)

    If oThemeManager.Themes.Item(i).Name <> oThemeManager.ActiveTheme.Name Then
        oThemeManager.Themes.Item(i).Activate
    Else
        Call MsgBox("The selected theme is active, please select another theme.", vbOKOnly, "Inventor")
    End If

End Sub
```

This sample demonstrates how to change the Inventor theme.

Copy Code

```

    Dim oThemeManager As ThemeManager
    oThemeManager = ThisApplication.ThemeManager

    Dim oTheme As Theme, i As Long: i = 1
    Dim sNames As String, sName As String
    For Each oTheme In oThemeManager.Themes
        sNames = sNames & i & "): " & oTheme.Name & vbCrLf
        i = i + 1
    Next

    sName = InputBox("Which theme would you like to set: " & vbCrLf & vbCrLf & sNames, "Inventor")
    i = CInt(sName)

    If oThemeManager.Themes.Item(i).Name <> oThemeManager.ActiveTheme.Name Then
        oThemeManager.Themes.Item(i).Activate
    Else
        Call MsgBox("The selected theme is active, please select another theme.", vbOKOnly, "Inventor")
    End If
```

UCS by three points

Description

This sample demonstrates the creation of a User Coordinate System (UCS) based on 3 points that define the origin, x-direction and y-direction for the UCS.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub CreateUCSBy3Points()
    ' Create a new part document
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject)

    ' Set a reference to the PartComponentDefinition object
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create 3 workpoints to define the origin, x-direction and y-direction points
    Dim oWorkPoint1 As WorkPoint
    Set oWorkPoint1 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(2, 0, 0))

    Dim oWorkPoint2 As WorkPoint
    Set oWorkPoint2 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(4, 0, 0))

    Dim oWorkPoint3 As WorkPoint
```

```
    Set oWorkPoint3 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(2, 2, 0))

    ' Create an empty definition object
    Dim oUCSDef As UserCoordinateSystemDefinition
    Set oUCSDef = oCompDef.UserCoordinateSystems.CreateDefinition

    ' Set it to be based on the 3 points
    Call oUCSDef.SetByThreePoints(oWorkPoint1, oWorkPoint2, oWorkPoint3)

    ' Create the UCS
    Dim oUCS As UserCoordinateSystem
    Set oUCS = oCompDef.UserCoordinateSystems.Add(oUCSDef)
End Sub
```

Copy Code

```
    ' Create a new part document
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject)

    ' Set a reference to the PartComponentDefinition object
    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create 3 workpoints to define the origin, x-direction and y-direction points
    Dim oWorkPoint1 As WorkPoint
    oWorkPoint1 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(2, 0, 0))

    Dim oWorkPoint2 As WorkPoint
    oWorkPoint2 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(4, 0, 0))

    Dim oWorkPoint3 As WorkPoint
    oWorkPoint3 = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(2, 2, 0))

    ' Create an empty definition object
    Dim oUCSDef As UserCoordinateSystemDefinition
    oUCSDef = oCompDef.UserCoordinateSystems.CreateDefinition

    ' Set it to be based on the 3 points
    Call oUCSDef.SetByThreePoints(oWorkPoint1, oWorkPoint2, oWorkPoint3)

    ' Create the UCS
    Dim oUCS As UserCoordinateSystem
    oUCS = oCompDef.UserCoordinateSystems.Add(oUCSDef)
```

UCS by transformation matrix

Description

This sample demonstrates the creation of a user coordinate system (UCS) by specifying a transformation matrix.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub CreateUCSByTransformationMatrix()
    ' Create a new part document
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject)

    ' Set a reference to the PartComponentDefinition object
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create an identity matrix
    Dim oMatrix As Matrix
    Set oMatrix = oTG.CreateMatrix

    ' Rotate about Z-Axis by 45 degrees
    Call oMatrix.SetToRotation(3.14159 / 4, oTG.CreateVector(0, 0, 1), oTG.CreatePoint(0, 0, 0))

    ' Translate the origin to (2, 2, 2)
    Dim oTranslationMatrix As Matrix
    Set oTranslationMatrix = oTG.CreateMatrix
    Call oTranslationMatrix.SetTranslation(oTG.CreateVector(2, 2, 2))

    Call oMatrix.TransformBy(oTranslationMatrix)

    ' Create an empty definition object
    Dim oUCSDef As UserCoordinateSystemDefinition
    Set oUCSDef = oCompDef.UserCoordinateSystems.CreateDefinition

    ' Set it to be based on the defined matrix
    oUCSDef.Transformation = oMatrix

    ' Create the UCS
    Dim oUCS As UserCoordinateSystem
```

```
    Set oUCS = oCompDef.UserCoordinateSystems.Add(oUCSDef)
End Sub
```

Copy Code

```
    ' Create a new part document
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject)

    ' Set a reference to the PartComponentDefinition object
    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create an identity matrix
    Dim oMatrix As Matrix
    oMatrix = oTG.CreateMatrix

    ' Rotate about Z-Axis by 45 degrees
    Call oMatrix.SetToRotation(3.14159 / 4, oTG.CreateVector(0, 0, 1), oTG.CreatePoint(0, 0, 0))

    ' Translate the origin to (2, 2, 2)
    Dim oTranslationMatrix As Matrix
    oTranslationMatrix = oTG.CreateMatrix
    Call oTranslationMatrix.SetTranslation(oTG.CreateVector(2, 2, 2))

    Call oMatrix.TransformBy(oTranslationMatrix)

    ' Create an empty definition object
    Dim oUCSDef As UserCoordinateSystemDefinition
    oUCSDef = oCompDef.UserCoordinateSystems.CreateDefinition

    ' Set it to be based on the defined matrix
    oUCSDef.Transformation = oMatrix

    ' Create the UCS
    Dim oUCS As UserCoordinateSystem
    oUCS = oCompDef.UserCoordinateSystems.Add(oUCSDef)
```

Display information about parameter tolerances.

Description

Dumps out information to the Immediate window about tolerance information associated with parameters. A part document must be active when this is run.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpParamInfo()
    ' Get the active document assuming it is a part.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Get the component definition.  This owns the part specific info for the part.
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Get the parameters.  This collection contains all parameters, regardless
    ' of how they were created.
    Dim params As Parameters
    Set params = partDef.Parameters

    ' Get the model parameters.  This collection contains the parameters that were
    ' created as a by-product of creating something in Inventor that's driven
    ' by a parameter (dimension constraint, feature, work feature, etc.).
    Dim modelParams As ModelParameters
    Set modelParams = params.ModelParameters

    ' Iterate through the model parameters.
    Dim modelParam As ModelParameter
    For Each modelParam In modelParams
        Call DisplayToleranceInfo(modelParam)
    Next
End Sub

' Utility function used to display tolerance info for a parameter.
Private Sub DisplayToleranceInfo(Param As Parameter)
    ' Get the Tolerance object from the parameter.
    Dim tol As Tolerance
    Set tol = Param.Tolerance

    ' Display some info about the tolerance.
    Select Case tol.ToleranceType
        Case kDefaultTolerance
            Debug.Print "kDefaultTolerance"
        Case kBasicTolerance
            Debug.Print "kBasicTolerance"
        Case kDeviationTolerance
            Debug.Print "kDeviationTolerance"
        Case kLimitLinearTolerance
            Debug.Print "kLimitLinearTolerance"
```

```
        Case kLimitsFitsLinearTolerance
            Debug.Print "kLimitsFitsLinearTolerance"
        Case kLimitsFitsShowSizeTolerance
            Debug.Print "kLimitsFitsShowSizeTolerance"
        Case kLimitsFitsShowTolerance
            Debug.Print "kLimitsFitsShowTolerance"
        Case kLimitsFitsStackedTolerance
            Debug.Print "kLimitsFitsStackedTolerance"
        Case kLimitsStackedTolerance
            Debug.Print "kLimitsStackedTolerance"
        Case kMaxTolerance
            Debug.Print "kMaxTolerance"
        Case kMinTolerance
            Debug.Print "kMinTolerance"
        Case kOverrideTolerance
            Debug.Print "kOverrideTolerance"
        Case kReferenceTolerance
            Debug.Print "kReferenceTolerance"
        Case kSymmetricTolerance
            Debug.Print "kSymmetricTolerance"
    End Select

    Debug.Print "  Upper: " & tol.Upper
    Debug.Print "  Lower: " & tol.Lower
    Debug.Print "  Hole: " & tol.HoleTolerance
    Debug.Print "  Shaft: " & tol.ShaftTolerance
End Sub
```

Copy Code

```
Sub Main
    ' Get the active document assuming it is a part.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    ' Get the component definition.  This owns the part specific info for the part.
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Get the parameters.  This collection contains all parameters, regardless
    ' of how they were created.
    Dim params As Parameters
    params = partDef.Parameters

    ' Get the model parameters.  This collection contains the parameters that were
    ' created as a by-product of creating something in Inventor that's driven
    ' by a parameter (dimension constraint, feature, work feature, etc.).
    Dim modelParams As ModelParameters
    modelParams = params.ModelParameters

    ' Iterate through the model parameters.
    Dim modelParam As ModelParameter
    For Each modelParam In modelParams
        Call DisplayToleranceInfo(modelParam)
    Next
End Sub

' Utility function used to display tolerance info for a parameter.
Private Sub DisplayToleranceInfo(Param As Parameter)
    ' Get the Tolerance object from the parameter.
    Dim tol As Tolerance
    tol = Param.Tolerance

    ' Display some info about the tolerance.
    Select Case tol.ToleranceType
        Case kDefaultTolerance
            Logger.Info("kDefaultTolerance")

        Case kBasicTolerance
            Logger.Info("kBasicTolerance")

        Case kDeviationTolerance
            Logger.Info("kDeviationTolerance")

        Case kLimitLinearTolerance
            Logger.Info("kLimitLinearTolerance")

        Case kLimitsFitsLinearTolerance
            Logger.Info("kLimitsFitsLinearTolerance")

        Case kLimitsFitsShowSizeTolerance
            Logger.Info("kLimitsFitsShowSizeTolerance")

        Case kLimitsFitsShowTolerance
            Logger.Info("kLimitsFitsShowTolerance")

        Case kLimitsFitsStackedTolerance
            Logger.Info("kLimitsFitsStackedTolerance")

        Case kLimitsStackedTolerance
            Logger.Info("kLimitsStackedTolerance")

        Case kMaxTolerance
            Logger.Info("kMaxTolerance")

        Case kMinTolerance
            Logger.Info("kMinTolerance")

        Case kOverrideTolerance
            Logger.Info("kOverrideTolerance")

        Case kReferenceTolerance
            Logger.Info("kReferenceTolerance")

        Case kSymmetricTolerance
            Logger.Info("kSymmetricTolerance")
```

```
    End Select

    Logger.Info("  Upper: " & tol.Upper)

    Logger.Info("  Lower: " & tol.Lower)

    Logger.Info("  Hole: " & tol.HoleTolerance)

    Logger.Info("  Shaft: " & tol.ShaftTolerance)

End Sub
```

Create user parameters

Description

This sample demonstrates creating user parameters using an expression and a value. A part document must be open and it must not contain user parameters named "NewParam1" and "NewParam2".

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateParameters()
    ' Get the active document.  Assumes a part document is active.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Get the UserParameters collection
    Dim userParams As UserParameters
    Set userParams = partDoc.ComponentDefinition.Parameters.UserParameters

    ' Create a parameter using an expression.  The parameters unit is specified
    ' as millimeters, but the value of the parameter will be 3 inches because
    ' the unit is specified as part of the expression.
    Dim param As Parameter
    Set param = userParams.AddByExpression("NewParam1", "3 in", kMillimeterLengthUnits)

    ' Create a parameter using a value.  When setting by value, the value is always
    ' in database units.  In this case it is a length so it will always be in
    ' centimeters.  The units used for the parameter will be the current length units
    ' of the document because it's defined to use the default display length units.
    Set param = userParams.AddByValue("NewParam2", 3 * 2.54, kDefaultDisplayLengthUnits)
End Sub
```

Copy Code

```
    ' Get the active document.  Assumes a part document is active.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    ' Get the UserParameters collection
    Dim userParams As UserParameters
    userParams = partDoc.ComponentDefinition.Parameters.UserParameters

    ' Create a parameter using an expression.  The parameters unit is specified
    ' as millimeters, but the value of the parameter will be 3 inches because
    ' the unit is specified as part of the expression.
    Dim param As Parameter
    param = userParams.AddByExpression("NewParam1", "3 in", kMillimeterLengthUnits)

    ' Create a parameter using a value.  When setting by value, the value is always
    ' in database units.  In this case it is a length so it will always be in
    ' centimeters.  The units used for the parameter will be the current length units
    ' of the document because it's defined to use the default display length units.
    param = userParams.AddByValue("NewParam2", 3 * 2.54, kDefaultDisplayLengthUnits)
```

Set the value of a parameter

Description

Sets the value of an existing parameter. A part must be open that contains a parameter named "Length".

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SetParameter()
    ' Get the Parameters object. Assumes a part or assembly document is active.
    Dim oParameters As Parameters
    Set oParameters = ThisApplication.ActiveDocument.ComponentDefinition.Parameters
```

```
    ' Get the parameter named "Length".
    Dim oLengthParam As Parameter
    Set oLengthParam = oParameters.Item("Length")

    ' Change the equation of the parameter.
    oLengthParam.Expression = "3.5 in"

    ' Update the document.
    ThisApplication.ActiveDocument.Update
End Sub
```

Copy Code

```
    ' Get the Parameters object. Assumes a part or assembly document is active.
    Dim oParameters As Parameters
    oParameters = ThisApplication.ActiveDocument.ComponentDefinition.Parameters

    ' Get the parameter named "Length".
    Dim oLengthParam As Parameter
    oLengthParam = oParameters.Item("Length")

    ' Change the equation of the parameter.
    oLengthParam.Expression = "3.5 in"

    ' Update the document.
    ThisApplication.ActiveDocument.Update
```

Text and Boolean user parameter creation

Description

This sample demonstrates how to create Text and Boolean user parameter.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateTextBooleanUserParameter()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oUserParameters As UserParameters
    Set oUserParameters = oDoc.ComponentDefinition.Parameters.UserParameters

    Dim oTextParam As UserParameter
    Set oTextParam = oUserParameters.AddByValue("Printer_Name", "Inventor_3D_Printer1", kTextUnits)

    Dim oBooleanParam As UserParameter
    Set oBooleanParam = oUserParameters.AddByValue("Auto_Print", True, kBooleanUnits)
End Sub
```

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oUserParameters As UserParameters
    oUserParameters = oDoc.ComponentDefinition.Parameters.UserParameters

    Dim oTextParam As UserParameter
    oTextParam = oUserParameters.AddByValue("Printer_Name", "Inventor_3D_Printer1", kTextUnits)

    Dim oBooleanParam As UserParameter
    oBooleanParam = oUserParameters.AddByValue("Auto_Print", True, kBooleanUnits)
```

Set active project

Description

The following sample demonstrates the activation of an Inventor project.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SetActiveProject()
    ' Check to make sure a document isn't open.
    If ThisApplication.Documents.Count > 0 Then
        MsgBox "All documents must be closed before changing the project."
        Exit Sub
```

```
    End If

    ' Set a reference to the DesignProjectManager object.
    Dim oDesignProjectMgr As DesignProjectManager
    Set oDesignProjectMgr = ThisApplication.DesignProjectManager

    ' Show the current project.
    Debug.Print "Old active project: " & oDesignProjectMgr.ActiveDesignProject.FullFileName

    ' Get the project to activate
    ' This assumes that "C:\Temp\MyProject.ipj" exists.
    Dim oProject As DesignProject
    Set oProject = oDesignProjectMgr.DesignProjects.ItemByName("C:\temp\MyProject.ipj")

    ' Activate the project
    oProject.Activate

    ' Show the current project after making the project change.
    Debug.Print "New active project: " & oDesignProjectMgr.ActiveDesignProject.FullFileName
End Sub
```

Copy Code

```
    ' Check to make sure a document isn't open.
    If ThisApplication.Documents.Count > 0 Then
        MsgBox("Please close all documents before editing Inventor project.",,"iLogic")
    Else
        ' Set a reference to the DesignProjectManager object.
        Dim oDesignProjectMgr As DesignProjectManager
        oDesignProjectMgr = ThisApplication.DesignProjectManager

        ' Show the current project.
        Logger.Info("Old active project: " & oDesignProjectMgr.ActiveDesignProject.FullFileName)

        ' Get the project to activate
        ' This assumes that "C:\Temp\MyProject.ipj" exists.
        Dim oProject As DesignProject
        oProject = oDesignProjectMgr.DesignProjects.ItemByName("C:\temp\MyProject.ipj")

        ' Activate the project
        oProject.Activate

        ' Show the current project after making the project change.
        Logger.Info("New active project: " & oDesignProjectMgr.ActiveDesignProject.FullFileName)
    End If
```

Create project

Description

The following sample demonstrates the creation of an Inventor project.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateProject()
    ' Set a reference to the DesignProjectManager object.
    Dim oDesignProjectMgr As DesignProjectManager
    Set oDesignProjectMgr = ThisApplication.DesignProjectManager

    ' Create a new singler user project
    Dim oProject As DesignProject
    Set oProject = oDesignProjectMgr.DesignProjects.Add(kSingleUserMode, "MyProject", "C:\temp\")
End Sub
```

Copy Code

```
    ' Set a reference to the DesignProjectManager object.
    Dim oDesignProjectMgr As DesignProjectManager
    oDesignProjectMgr = ThisApplication.DesignProjectManager

    If ThisApplication.Documents.Count > 0 Then
        MsgBox("Please close all documents before creating new Inventor project.",,"iLogic")
    Else
         ' Create a new singler user project
        Dim oProject As DesignProject
        oProject = oDesignProjectMgr.DesignProjects.Add(kSingleUserMode, "MyProject", "C:\temp\")
    End If
```

Query and create library paths

Description

The following sample demonstrates querying existing library paths associated with a project and adding a new library path.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub QueryAndCreateLibraryPaths()
    Dim oProjectMgr As DesignProjectManager
    Set oProjectMgr = ThisApplication.DesignProjectManager

    ' Get the active project
    Dim oProject As DesignProject
    Set oProject = oProjectMgr.ActiveDesignProject

    Dim oLibraryPaths As ProjectPaths
    Set oLibraryPaths = oProject.LibraryPaths

    ' Add a new library path to the end of the list
    Call oLibraryPaths.Add("MyLibraryPath", "C:\temp\MyLibrary")

    ' Print all library names and paths
    Dim oLibraryPath As ProjectPath
    For Each oLibraryPath In oLibraryPaths
        Debug.Print "Name: " & oLibraryPath.Name & " Path: " & oLibraryPath.Path
    Next
End Sub
```

Copy Code

```
    Dim oProjectMgr As DesignProjectManager
    oProjectMgr = ThisApplication.DesignProjectManager

    ' Get the active project
    Dim oProject As DesignProject
    oProject = oProjectMgr.ActiveDesignProject

    Dim oLibraryPaths As ProjectPaths
    oLibraryPaths = oProject.LibraryPaths

    If ThisApplication.Documents.Count > 0 Then
        MsgBox("Please close all documents before editing Inventor project.",,"iLogic")
    Else
        ' Add a new library path to the end of the list
        Call oLibraryPaths.Add("MyLibraryPath", "C:\temp\MyLibrary")

        ' Print all library names and paths
        Dim oLibraryPath As ProjectPath
        For Each oLibraryPath In oLibraryPaths
            Logger.Info("Name: " & oLibraryPath.Name & " Path: " & oLibraryPath.Path)
        Next
    End If
```

Change color of part, features or faces

Description

This sample demonstrates how to use MiniToolBar to change appearance color of part or features or faces.

Code Samples

- VBA
- iLogic

You can either run below sample with an open part document or it will create a new part document.

Copy Code

```
' This sample demonstrates how to change colors using mini-toolbar
Sub ChangeAppearanceMiniToolbarSample()
    Dim oDoc As PartDocument
    If Not (ThisApplication.ActiveDocument Is Nothing) Then
        If ThisApplication.ActiveDocument.DocumentType = kPartDocumentObject Then
            If (MsgBox("Would you like to create new appearance assets with active document?", vbYesNo, "Autodesk Inventor Prompt") =
                Set oDoc = ThisApplication.ActiveDocument
            Else
                Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

                ' Create a block
                Call CreateSolids(oDoc)
            End If
        End If
    Else
        Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

        ' Create a block
        Call CreateSolids(oDoc)
    End If

    ' Create seven appearances for use
    Call CreateColors(oDoc)
    Dim oCamera As Camera
    Set oCamera = oDoc.Views(1).Camera

    oCamera.Fit
```

```
    oCamera.Apply

    Dim oMiniToolbarEvents As New clsColorChange

    Call oMiniToolbarEvents.Init(oDoc)

End Sub

Private Sub CreateSolids(oDoc As PartDocument)
    Dim oSk As PlanarSketch
    Set oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    Call oSk.SketchLines.AddAsTwoPointRectangle(ThisApplication.TransientGeometry.CreatePoint2d(-2, -2), ThisApplication.TransientGeom

    Dim oProfile As Profile
    Set oProfile = oSk.Profiles.AddForSolid

    Dim oBlockDef As ExtrudeDefinition
    Set oBlockDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oBlockDef.SetDistanceExtent(4, kSymmetricExtentDirection)
    Dim oBlock As ExtrudeFeature
    Set oBlock = oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oBlockDef)

    Set oSk = oDoc.ComponentDefinition.Sketches.Add(oBlock.EndFaces(1))
    Call oSk.SketchCircles.AddByCenterRadius(ThisApplication.TransientGeometry.CreatePoint2d(12, 12), 2)

    Set oProfile = oSk.Profiles.AddForSolid
    Dim oCylinderDef As ExtrudeDefinition, oCylinder As ExtrudeFeature
    Set oCylinderDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oCylinderDef.SetDistanceExtent(4, kSymmetricExtentDirection)
    Set oCylinder = oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oCylinderDef)
End Sub

Private Sub CreateColors(oDoc As PartDocument)
    Dim bCreateNewAppearance As Boolean: bCreateNewAppearance = False

    Dim oCreateColors As Transaction
    Set oCreateColors = ThisApplication.TransactionManager.StartTransaction(oDoc, "Create custom colors")
        On Error Resume Next
        Dim oAsset As Asset, oColor As Color
        Set oAsset = oDoc.Assets.Item("Red")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Red", "Red")
            ' Add Red
            Set oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0)
            oAsset.Item("generic_diffuse").Value = oColor
            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Orange")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Orange", "Orange")
            ' Add Orange
            Set oColor = ThisApplication.TransientObjects.CreateColor(255, 165, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Yellow")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Yellow", "Yellow")
            ' Add Yellow
            Set oColor = ThisApplication.TransientObjects.CreateColor(255, 255, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Green")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Green", "Green")
            ' Add Green
            Set oColor = ThisApplication.TransientObjects.CreateColor(0, 255, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Blue")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Blue", "Blue")
            ' Add Blue
            Set oColor = ThisApplication.TransientObjects.CreateColor(0, 0, 255)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Indigo")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Indigo", "Indigo")
            ' Add Indigo
            Set oColor = ThisApplication.TransientObjects.CreateColor(75, 0, 130)
            oAsset.Item("generic_diffuse").Value = oColor
            bCreateNewAppearance = True
            Err.Clear
        End If

        Set oAsset = oDoc.Assets.Item("Purple")
        If Err Then
            Set oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Purple", "Purple")
```

```
            ' Add Purple
            Set oColor = ThisApplication.TransientObjects.CreateColor(160, 32, 240)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        If bCreateNewAppearance Then
            oCreateColors.End
        Else
            oCreateColors.Abort
        End If

End Sub

'**********************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsColorChange".  The name
' can be changed but you'll need to change the declaration in
' the calling function "ChangeAppearanceMiniToolbarSample" to use the new name.
Private WithEvents m_MiniToolbar As MiniToolbar
Private WithEvents m_Colors As MiniToolbarComboBox
Private WithEvents m_Filter As MiniToolbarDropdown
Private WithEvents m_Preview As MiniToolbarCheckBox
Private m_PreviewColor As MiniToolbarCheckBox

Private WithEvents oInteractionEvents As InteractionEvents
Private WithEvents m_SelectEvents As SelectEvents
Private m_ChangeColorTransaction As Transaction

Private m_Doc As PartDocument
Private m_DefaultColor As Asset

Private bIsinteractionStarted As Boolean

Private bNeedTransaction As Boolean
Private bStop As Boolean

Public Sub Init(oDoc As PartDocument)
    Set m_Doc = oDoc
    Set m_DefaultColor = m_Doc.ActiveAppearance

    ' Create interaction events
    Set oInteractionEvents = ThisApplication.CommandManager.CreateInteractionEvents
    'oInteractionEvents.InteractionDisabled = False

    Set m_SelectEvents = oInteractionEvents.SelectEvents
    'm_SelectEvents.ClearSelectionFilter
    'm_SelectEvents.SingleSelectEnabled = False
    'm_SelectEvents.Enabled = True

    ' Create mini-tool bar for changing appearance
    Set m_MiniToolbar = oInteractionEvents.CreateMiniToolbar
    Call InitiateMiniToolbar

    bStop = False
    Set m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")

    Do
        ThisApplication.UserInterfaceManager.DoEvents
    Loop Until bStop

End Sub

Private Sub InitiateMiniToolbar()
    m_MiniToolbar.ShowOK = True
    m_MiniToolbar.ShowApply = True
    m_MiniToolbar.ShowCancel = True

    Dim oControls As MiniToolbarControls
    Set oControls = m_MiniToolbar.Controls
    oControls.Item("MTB_Options").Visible = False

    Set m_Filter = m_MiniToolbar.Controls.AddDropdown("Filter", False, True, True, False)
    Call m_Filter.AddItem("Part", "Part", "Filter_Part", False, False)
    Call m_Filter.AddItem("Feature", "Feature", "Filter_Feature", False, False)
    Call m_Filter.AddItem("Face", "Face", "Filter_Face", False, False)


    Set m_Colors = oControls.AddComboBox("Colors", True, True, 50)
    Call m_Colors.AddItem("Default", "Use default color", "Default", False)
    Call m_Colors.AddItem("Red", "Red", "Red", False)
    Call m_Colors.AddItem("Orange", "Orange", "Orange", False)
    Call m_Colors.AddItem("Yellow", "Yellow", "Yellow", False)
    Call m_Colors.AddItem("Green", "Green", "Green", False)
    Call m_Colors.AddItem("Blue", "Blue", "Blue", False)
    Call m_Colors.AddItem("Indigo", "Indigo", "Indigo", False)
    Call m_Colors.AddItem("Purple", "Purple", "Purple", False)

    oControls.AddNewLine

    ' Specify if preview the color when hover a color item
    Set m_PreviewColor = m_MiniToolbar.Controls.AddCheckBox("PreviewColor", "Hover color preview", "Whether preview color when hover o

    ' Position the mini-tool bar to the top-left.
    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)

    m_MiniToolbar.Visible = True
    m_MiniToolbar.Position = oPosition
End Sub

Private Sub m_Colors_OnItemHoverStart(ByVal ListItem As MiniToolbarListItem)
    ' Preview the color when hover on it.
    If m_PreviewColor.Checked Then
        Call ChangeColor(ListItem.Text)
```

```
    End If
End Sub

Private Sub m_Colors_OnSelect(ByVal ListItem As MiniToolbarListItem)
    ' Check if the selected color is already used for the part/objects
    If m_Filter.SelectedItem.Text = "Part" Then
        If m_Doc.ActiveAppearance.Name = ListItem.Text Then
            bNeedTransaction = False
        Else
            bNeedTransaction = True
        End If
    Else
        bNeedTransaction = True
    End If

    Call ChangeColor(ListItem.Text)

End Sub
' Change filter for assigning color
Private Sub m_Filter_OnSelect(ByVal ListItem As MiniToolbarListItem)
    If ThisApplication.TransactionManager.CurrentTransaction.DisplayName = "Change Appearance" Then
        ThisApplication.TransactionManager.CurrentTransaction.Abort
    End If

    Set m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")

    Select Case ListItem.Text
        Case "Part"
            m_Doc.SelectSet.Clear
            m_SelectEvents.ResetSelections
            m_SelectEvents.ClearSelectionFilter
            m_SelectEvents.AddSelectionFilter kPartDefaultFilter
            oInteractionEvents.SetCursor kCursorTypeDefault
        Case "Feature"
            m_Doc.SelectSet.Clear

            m_SelectEvents.ResetSelections
            m_SelectEvents.ClearSelectionFilter
            m_SelectEvents.AddSelectionFilter kPartFeatureFilter

            If Not bIsinteractionStarted Then
                oInteractionEvents.Start
                bIsinteractionStarted = True
            End If
        Case "Face"
            m_Doc.SelectSet.Clear
            m_SelectEvents.ResetSelections
            m_SelectEvents.ClearSelectionFilter
            m_SelectEvents.AddSelectionFilter kPartFaceFilter
            If Not bIsinteractionStarted Then
                oInteractionEvents.Start
                bIsinteractionStarted = True
            End If

    End Select
    m_Doc.Views(1).Update
    Call ChangeColor(ListItem.Text)

End Sub

Private Sub m_MiniToolbar_OnApply()

    If (m_Filter.SelectedItem.Text = "Feature" Or m_Filter.SelectedItem.Text = "Face") And (m_SelectEvents.SelectedEntities.Count = 0)
        m_ChangeColorTransaction.Abort
        m_Doc.Views(1).Update
        Set m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")
        Exit Sub
    Else
        If bNeedTransaction Then ' Change color style
            Call ChangeColor(m_Colors.SelectedItem.Text)
            m_ChangeColorTransaction.End
        Else ' If no change to the color style
            m_ChangeColorTransaction.Abort
        End If

        ' Clear current selection for Feature and Face filter.
        If (m_Filter.SelectedItem.Text = "Feature" Or m_Filter.SelectedItem.Text = "Face") Then
            m_Doc.SelectSet.Clear
            m_SelectEvents.ResetSelections
        End If
    End If

    Set m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")
End Sub

Private Sub m_MiniToolbar_OnCancel()
    bStop = True
    If ThisApplication.TransactionManager.CurrentTransaction Is m_ChangeColorTransaction Then
        m_ChangeColorTransaction.Abort
    End If
    m_SelectEvents.AddSelectionFilter kPartDefaultFilter
    If bIsinteractionStarted Then oInteractionEvents.Stop
    m_Doc.Views(1).Update

End Sub

Private Sub m_MiniToolbar_OnOK()
    bStop = True
    If bNeedTransaction Then ' Change color
        Call ChangeColor(m_Colors.SelectedItem.Text)
        m_ChangeColorTransaction.End
    Else ' If no change to the color style
        m_ChangeColorTransaction.Abort
    End If
End Sub

Private Sub oInteractionEvents_OnTerminate()
```

```
    If ThisApplication.TransactionManager.CurrentTransaction Is m_ChangeColorTransaction Then
        m_ChangeColorTransaction.Abort
    End If
    If bIsinteractionStarted Then
        oInteractionEvents.Stop
    End If
    m_Doc.Views(1).Update
End Sub

Private Sub ChangeColor(sColor As String)
        Debug.Print "Passed in:" & sColor
        If m_Filter.SelectedItem.Text = "Part" Then
            Select Case sColor
                Case "Default"
                    m_Doc.ActiveAppearance = m_DefaultColor
                Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                    m_Doc.ActiveAppearance = m_Doc.AppearanceAssets.Item(sColor)
            End Select
        ElseIf m_Filter.SelectedItem.Text = "Feature" Then
            If m_SelectEvents.SelectedEntities.Count Then
                Dim oFeature As PartFeature, oSelectedObj As Object

                For Each oSelectedObj In m_SelectEvents.SelectedEntities
                    If InStr(1, TypeName(oSelectedObj), "Feature") Then
                        Set oFeature = oSelectedObj

                        Select Case sColor 'm_Colors.SelectedItem.Text
                            Case "Default"
                                oFeature.Appearance = m_DefaultColor
                            Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                                oFeature.Appearance = m_Doc.AppearanceAssets.Item(sColor)
                        End Select
                    End If
                Next
            End If
        ElseIf m_Filter.SelectedItem.Text = "Face" Then
            If m_SelectEvents.SelectedEntities.Count Then
                Dim oFace As Face

                For Each oSelectedObj In m_SelectEvents.SelectedEntities
                    If InStr(1, TypeName(oSelectedObj), "Face") Then
                        Set oFace = oSelectedObj

                        Select Case sColor
                            Case "Default"
                                oFace.Appearance = m_DefaultColor
                            Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                                oFace.Appearance = m_Doc.AppearanceAssets.Item(sColor)
                        End Select
                    End If
                Next
            End If
        End If
End Sub
```

You can either run below sample with an open part document or it will create a new part document.

Copy Code

```
' This sample demonstrates how to change colors using mini-toolbar
Sub Main
    Dim oDoc As PartDocument
    If Not (ThisApplication.ActiveDocument Is Nothing) Then
        If ThisApplication.ActiveDocument.DocumentType = kPartDocumentObject Then
            If (MsgBox("Would you like to create new appearance assets with active document?", vbYesNo, "Autodesk Inventor Prompt") =
                oDoc = ThisApplication.ActiveDocument
            Else
                oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

                ' Create a block
                Call CreateSolids(oDoc)
            End If
        End If
    Else
        oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

        ' Create a block
        Call CreateSolids(oDoc)
    End If

    ' Create seven appearances for use
    Call CreateColors(oDoc)
    Dim oCamera As Camera
    oCamera = oDoc.Views(1).Camera

    oCamera.Fit
    oCamera.Apply

    Dim oMiniToolbarEvents As New clsColorChange

    Call oMiniToolbarEvents.Init(oDoc, ThisApplication, Logger)

End Sub

Private Sub CreateSolids(oDoc As PartDocument)
    Dim oSk As PlanarSketch
    oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    Call oSk.SketchLines.AddAsTwoPointRectangle(ThisApplication.TransientGeometry.CreatePoint2d(-2, -2), ThisApplication.TransientGeom

    Dim oProfile As Profile
    oProfile = oSk.Profiles.AddForSolid

    Dim oBlockDef As ExtrudeDefinition
    oBlockDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oBlockDef.SetDistanceExtent(4, kSymmetricExtentDirection)
    Dim oBlock As ExtrudeFeature
```

```
    oBlock = oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oBlockDef)

    oSk = oDoc.ComponentDefinition.Sketches.Add(oBlock.EndFaces(1))
    Call oSk.SketchCircles.AddByCenterRadius(ThisApplication.TransientGeometry.CreatePoint2d(12, 12), 2)

    oProfile = oSk.Profiles.AddForSolid
    Dim oCylinderDef As ExtrudeDefinition, oCylinder As ExtrudeFeature
    oCylinderDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oCylinderDef.SetDistanceExtent(4, kSymmetricExtentDirection)
    oCylinder = oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oCylinderDef)
End Sub

Private Sub CreateColors(oDoc As PartDocument)
    Dim bCreateNewAppearance As Boolean: bCreateNewAppearance = False

    Dim oCreateColors As Transaction
    oCreateColors = ThisApplication.TransactionManager.StartTransaction(oDoc, "Create custom colors")
        On Error Resume Next
        Dim oAsset As Asset, oColor As Color
        oAsset = oDoc.Assets.Item("Red")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Red", "Red")
            ' Add Red
            oColor = ThisApplication.TransientObjects.CreateColor(255, 0, 0)
            oAsset.Item("generic_diffuse").Value = oColor
            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Orange")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Orange", "Orange")
            ' Add Orange
            oColor = ThisApplication.TransientObjects.CreateColor(255, 165, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Yellow")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Yellow", "Yellow")
            ' Add Yellow
            oColor = ThisApplication.TransientObjects.CreateColor(255, 255, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Green")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Green", "Green")
            ' Add Green
            oColor = ThisApplication.TransientObjects.CreateColor(0, 255, 0)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Blue")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Blue", "Blue")
            ' Add Blue
            oColor = ThisApplication.TransientObjects.CreateColor(0, 0, 255)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Indigo")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Indigo", "Indigo")
            ' Add Indigo
            oColor = ThisApplication.TransientObjects.CreateColor(75, 0, 130)
            oAsset.Item("generic_diffuse").Value = oColor
            bCreateNewAppearance = True
            Err.Clear
        End If

        oAsset = oDoc.Assets.Item("Purple")
        If Err.Number > 0 Then
            oAsset = oDoc.Assets.Add(kAssetTypeAppearance, "Generic", "Purple", "Purple")
            ' Add Purple
            oColor = ThisApplication.TransientObjects.CreateColor(160, 32, 240)
            oAsset.Item("generic_diffuse").Value = oColor

            bCreateNewAppearance = True
            Err.Clear
        End If

        If bCreateNewAppearance Then
            oCreateColors.End
        Else
            oCreateColors.Abort
        End If

End Sub

Class clsColorChange
        Private ThisApplication As Inventor.Application
        Private Logger As Object
        Private WithEvents m_MiniToolbar As MiniToolbar
        Private WithEvents m_Colors As MiniToolbarComboBox
```

```
Private WithEvents m_Filter As MiniToolbarDropdown
Private WithEvents m_Preview As MiniToolbarCheckBox
Private m_PreviewColor As MiniToolbarCheckBox

Private WithEvents oInteractionEvents As InteractionEvents
Private WithEvents m_SelectEvents As SelectEvents
Private m_ChangeColorTransaction As Transaction

Private m_Doc As PartDocument
Private m_DefaultColor As Asset

Private bIsinteractionStarted As Boolean

Private bNeedTransaction As Boolean
Private bStop As Boolean

Public Sub Init(oDoc As PartDocument, oApp As Inventor.Application, oLogger As Object)
	ThisApplication = oApp
	Logger = oLogger

	m_Doc = oDoc
	m_DefaultColor = m_Doc.ActiveAppearance

	' Create interaction events
	oInteractionEvents = ThisApplication.CommandManager.CreateInteractionEvents
	'oInteractionEvents.InteractionDisabled = False

	m_SelectEvents = oInteractionEvents.SelectEvents
	'm_SelectEvents.ClearSelectionFilter
	'm_SelectEvents.SingleSelectEnabled = False
	'm_SelectEvents.Enabled = True

	' Create mini-tool bar for changing appearance
	m_MiniToolbar = oInteractionEvents.CreateMiniToolbar
	Call InitiateMiniToolbar

	bStop = False
	m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")

	Do
		ThisApplication.UserInterfaceManager.DoEvents
	Loop Until bStop

End Sub

Private Sub InitiateMiniToolbar()
	m_MiniToolbar.ShowOK = True
	m_MiniToolbar.ShowApply = True
	m_MiniToolbar.ShowCancel = True

	Dim oControls As MiniToolbarControls
	oControls = m_MiniToolbar.Controls
	oControls.Item("MTB_Options").Visible = False

	m_Filter = m_MiniToolbar.Controls.AddDropdown("Filter", False, True, True, False)
	Call m_Filter.AddItem("Part", "Part", "Filter_Part", False, False)
	Call m_Filter.AddItem("Feature", "Feature", "Filter_Feature", False, False)
	Call m_Filter.AddItem("Face", "Face", "Filter_Face", False, False)


	m_Colors = oControls.AddComboBox("Colors", True, True, 50)
	Call m_Colors.AddItem("Default", "Use default color", "Default", False)
	Call m_Colors.AddItem("Red", "Red", "Red", False)
	Call m_Colors.AddItem("Orange", "Orange", "Orange", False)
	Call m_Colors.AddItem("Yellow", "Yellow", "Yellow", False)
	Call m_Colors.AddItem("Green", "Green", "Green", False)
	Call m_Colors.AddItem("Blue", "Blue", "Blue", False)
	Call m_Colors.AddItem("Indigo", "Indigo", "Indigo", False)
	Call m_Colors.AddItem("Purple", "Purple", "Purple", False)

	oControls.AddNewLine

	' Specify if preview the color when hover a color item
	m_PreviewColor = m_MiniToolbar.Controls.AddCheckBox("PreviewColor", "Hover color preview", "Whether preview color when

	' Position the mini-tool bar to the top-left.
	Dim oPosition As Point2d
	oPosition = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)

	m_MiniToolbar.Visible = True
	m_MiniToolbar.Position = oPosition
End Sub

Private Sub m_Colors_OnItemHoverStart(ByVal ListItem As MiniToolbarListItem)
	' Preview the color when hover on it.
	If m_PreviewColor.Checked Then
		Call ChangeColor(ListItem.Text)
	End If
End Sub

Private Sub m_Colors_OnSelect(ByVal ListItem As MiniToolbarListItem)
	' Check if the selected color is already used for the part/objects
	If m_Filter.SelectedItem.Text = "Part" Then
		If m_Doc.ActiveAppearance.Name = ListItem.Text Then
			bNeedTransaction = False
		Else
			bNeedTransaction = True
		End If
	Else
		bNeedTransaction = True
	End If

	Call ChangeColor(ListItem.Text)

End Sub
' Change filter for assigning color
Private Sub m_Filter_OnSelect(ByVal ListItem As MiniToolbarListItem)
```

```
        If ThisApplication.TransactionManager.CurrentTransaction.DisplayName = "Change Appearance" Then
                ThisApplication.TransactionManager.CurrentTransaction.Abort
        End If

        m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")

        Select Case ListItem.Text
                Case "Part"
                        m_Doc.SelectSet.Clear
                        m_SelectEvents.ResetSelections
                        m_SelectEvents.ClearSelectionFilter
                        m_SelectEvents.AddSelectionFilter(kPartDefaultFilter)
                        oInteractionEvents.SetCursor(kCursorTypeDefault)
                Case "Feature"
                        m_Doc.SelectSet.Clear

                        m_SelectEvents.ResetSelections
                        m_SelectEvents.ClearSelectionFilter
                        m_SelectEvents.AddSelectionFilter(kPartFeatureFilter)

                        If Not bIsinteractionStarted Then
                                oInteractionEvents.Start
                                bIsinteractionStarted = True
                        End If
                Case "Face"
                        m_Doc.SelectSet.Clear
                        m_SelectEvents.ResetSelections
                        m_SelectEvents.ClearSelectionFilter
                        m_SelectEvents.AddSelectionFilter(kPartFaceFilter)
                        If Not bIsinteractionStarted Then
                                oInteractionEvents.Start
                                bIsinteractionStarted = True
                        End If

        End Select
        m_Doc.Views(1).Update
        Call ChangeColor(ListItem.Text)

End Sub

Private Sub m_MiniToolbar_OnApply()

        If (m_Filter.SelectedItem.Text = "Feature" Or m_Filter.SelectedItem.Text = "Face") And (m_SelectEvents.SelectedEntitie
                m_ChangeColorTransaction.Abort
                m_Doc.Views(1).Update
                m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")
                Exit Sub
        Else
                If bNeedTransaction Then ' Change color style
                        Call ChangeColor(m_Colors.SelectedItem.Text)
                        m_ChangeColorTransaction.End
                Else ' If no change to the color style
                        m_ChangeColorTransaction.Abort
                End If

                ' Clear current selection for Feature and Face filter.
                If (m_Filter.SelectedItem.Text = "Feature" Or m_Filter.SelectedItem.Text = "Face") Then
                        m_Doc.SelectSet.Clear
                        m_SelectEvents.ResetSelections
                End If
        End If

        m_ChangeColorTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Change Appearance")
End Sub

Private Sub m_MiniToolbar_OnCancel()
        bStop = True
        If ThisApplication.TransactionManager.CurrentTransaction Is m_ChangeColorTransaction Then
                m_ChangeColorTransaction.Abort
        End If
        m_SelectEvents.AddSelectionFilter(kPartDefaultFilter)
        If bIsinteractionStarted Then oInteractionEvents.Stop
        m_Doc.Views(1).Update

End Sub

Private Sub m_MiniToolbar_OnOK()
        bStop = True
        If bNeedTransaction Then ' Change color
                Call ChangeColor(m_Colors.SelectedItem.Text)
                m_ChangeColorTransaction.End
        Else ' If no change to the color style
                m_ChangeColorTransaction.Abort
        End If
End Sub

Private Sub oInteractionEvents_OnTerminate()

        If ThisApplication.TransactionManager.CurrentTransaction Is m_ChangeColorTransaction Then
                m_ChangeColorTransaction.Abort
        End If
        If bIsinteractionStarted Then
                oInteractionEvents.Stop
        End If
        m_Doc.Views(1).Update
End Sub

Private Sub ChangeColor(sColor As String)
                Logger.Info("Passed in:" & sColor)

                If m_Filter.SelectedItem.Text = "Part" Then
                        Select Case sColor
                                Case "Default"
                                        m_Doc.ActiveAppearance = m_DefaultColor
                                Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                                        m_Doc.ActiveAppearance = m_Doc.AppearanceAssets.Item(sColor)
                        End Select
```

```
                        ElseIf m_Filter.SelectedItem.Text = "Feature" Then
                                If m_SelectEvents.SelectedEntities.Count Then
                                        Dim oFeature As PartFeature, oSelectedObj As Object

                                        For Each oSelectedObj In m_SelectEvents.SelectedEntities
                                                If InStr(1, TypeName(oSelectedObj), "Feature") Then
                                                        oFeature = oSelectedObj

                                                        Select Case sColor 'm_Colors.SelectedItem.Text
                                                                Case "Default"
                                                                        oFeature.Appearance = m_DefaultColor
                                                                Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                                                                        oFeature.Appearance = m_Doc.AppearanceAssets.Item(sColor)
                                                        End Select
                                                End If
                                        Next
                                End If
                        ElseIf m_Filter.SelectedItem.Text = "Face" Then
                                If m_SelectEvents.SelectedEntities.Count Then
                                        Dim oFace As Face

                                        For Each oSelectedObj In m_SelectEvents.SelectedEntities
                                                If InStr(1, TypeName(oSelectedObj), "Face") Then
                                                        oFace = oSelectedObj

                                                        Select Case sColor
                                                                Case "Default"
                                                                        oFace.Appearance = m_DefaultColor
                                                                Case "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", "Purple"
                                                                        oFace.Appearance = m_Doc.AppearanceAssets.Item(sColor)
                                                        End Select
                                                End If
                                        Next
                                End If
                        End If
        End Sub
End Class
```

Create a simple appearance.

Description

Creates a sample appearance in the active part or assembly document.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSimpleColorAppearance()
    Dim doc As Document
    Set doc = ThisApplication.ActiveDocument

    ' Only document appearances can be edited, so that's what's created.
    ' This assumes a part or assembly document is active.
    Dim docAssets As Assets
    Set docAssets = doc.Assets

    ' Create a new appearance asset.
    Dim appearance As Asset
    Set appearance = docAssets.Add(kAssetTypeAppearance, "Generic", _
                                   "MyShinyRed", "My Shiny Red Color")

    Dim tobjs As TransientObjects
    Set tobjs = ThisApplication.TransientObjects


    Dim color As ColorAssetValue
    Set color = appearance.Item("generic_diffuse")
    color.value = tobjs.CreateColor(255, 15, 15)

    Dim floatValue As FloatAssetValue
    Set floatValue = appearance.Item("generic_reflectivity_at_0deg")
    floatValue.value = 0.5

    Set floatValue = appearance.Item("generic_reflectivity_at_90deg")
    floatValue.value = 0.5
End Sub
```

Copy Code

```
    Dim doc As Document
    doc = ThisApplication.ActiveDocument

    ' Only document appearances can be edited, so that's what's created.
    ' This assumes a part or assembly document is active.
    Dim docAssets As Assets
    docAssets = doc.Assets

    ' Create a new appearance asset.
    Dim appearance As Asset
    appearance = docAssets.Add(kAssetTypeAppearance, "Generic", _
                                   "MyShinyRed", "My Shiny Red Color")

    Dim tobjs As TransientObjects
```

```
    tobjs = ThisApplication.TransientObjects


    Dim color As ColorAssetValue
    color = appearance.Item("generic_diffuse")
    color.value = tobjs.CreateColor(255, 15, 15)

    Dim floatValue As FloatAssetValue
    floatValue = appearance.Item("generic_reflectivity_at_0deg")
    floatValue.value = 0.5

    floatValue = appearance.Item("generic_reflectivity_at_90deg")
    floatValue.value = 0.5
```

Write out all appearance information to a file.

Description

This sample writes out information about all of the appearances in all libraries. This can be useful when trying to use the API to modify existing appearances by allowing to easily see what information is available for an appearance.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpAllAppearancesInAllLibraries()
    ' Open a file to write the results.
    Open "C:\Temp\AllLibAppearanceDump.txt" For Output As #1

    ' Iterate through the libraries.
    Dim assetLib As AssetLibrary
    For Each assetLib In ThisApplication.AssetLibraries
        Print #1, "Library" & assetLib.DisplayName
        Print #1, "  DisplayName: " & assetLib.DisplayName
        Print #1, "  FullFileName: " & assetLib.FullFileName
        Print #1, "  InternalName: " & assetLib.InternalName
        Print #1, "  IsReadOnly: " & assetLib.IsReadOnly

        Dim appearance As Asset
        For Each appearance In assetLib.AppearanceAssets
            Print #1, "    Appearance"
            Print #1, "      DisplayName: " & appearance.DisplayName
            Print #1, "      Category: " & appearance.Category.DisplayName
            Print #1, "      HasTexture: " & appearance.HasTexture
            Print #1, "      IsReadOnly: " & appearance.IsReadOnly
            Print #1, "      Name: " & appearance.Name

            Dim value As AssetValue
            For Each value In appearance
                Call PrintAssetValue(value, 8)
            Next
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\AllLibAppearanceDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
```

```
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
            Else
                Print #1, indentChars & "  Colors"

                Dim colors() As color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub


' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
        Dim oWriter As System.IO.StreamWriter
```

```
Sub Main
    ' Open a file to write the results.
        oWriter = New StreamWriter("C:\Temp\AllLibAppearanceDump.txt")

    ' Iterate through the libraries.
    Dim assetLib As AssetLibrary
    For Each assetLib In ThisApplication.AssetLibraries
        oWriter.WriteLine("Library" & assetLib.DisplayName)
        oWriter.WriteLine("  DisplayName: " & assetLib.DisplayName)
        oWriter.WriteLine("  FullFileName: " & assetLib.FullFileName)
        oWriter.WriteLine("  InternalName: " & assetLib.InternalName)
        oWriter.WriteLine("  IsReadOnly: " & assetLib.IsReadOnly)

        Dim appearance As Asset
        For Each appearance In assetLib.AppearanceAssets
            oWriter.WriteLine("    Appearance")
            oWriter.WriteLine("      DisplayName: " & appearance.DisplayName)
            oWriter.WriteLine("      Category: " & appearance.Category.DisplayName)
            oWriter.WriteLine("      HasTexture: " & appearance.HasTexture)
            oWriter.WriteLine("      IsReadOnly: " & appearance.IsReadOnly)
            oWriter.WriteLine("      Name: " & appearance.Name)

            Dim value As AssetValue
            For Each value In appearance
                Call PrintAssetValue(value, 8)
            Next
        Next
    Next

        oWriter.Close()

    MsgBox("Finished writing output to ""C:\Temp\AllLibAppearanceDump.txt""")
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    oWriter.WriteLine(indentChars & "Value")
    oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
    oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
    oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            oWriter.WriteLine(indentChars & "  Type: Texture")

            Dim textureValue As TextureAssetValue
            textureValue = InValue

            Dim texture As AssetTexture
            texture = textureValue.Value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
                Case kTextureTypeChecker
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
                Case kTextureTypeGradient
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
                Case kTextureTypeMarble
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
                Case kTextureTypeNoise
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
                Case kTextureTypeSpeckle
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
                Case kTextureTypeTile
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
                Case kTextureTypeUnknown
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
                Case kTextureTypeWave
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
                Case kTextureTypeWood
                    oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
                Case Else
                    oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
            End Select

            oWriter.WriteLine(indentChars & "  Values")
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            oWriter.WriteLine(indentChars & "  Type: Boolean")

            Dim booleanValue As BooleanAssetValue
            booleanValue = InValue

            oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
        Case kAssetValueTypeChoice
            oWriter.WriteLine(indentChars & "  Type: Choice")

            Dim choiceValue As ChoiceAssetValue
            choiceValue = InValue

            oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

            Dim names() As String = New String() {}
            Dim choices() As String = New String() {}
            Call choiceValue.GetChoices(names, choices)
            oWriter.WriteLine(indentChars & "    Choices:")
            Dim i As Integer
            For i = 0 To UBound(names)
                oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
```

```
                Next
            Case kAssetValueTypeColor
                oWriter.WriteLine(indentChars & "  Type: Color")

                Dim colorValue As ColorAssetValue
                colorValue = InValue

                oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
                oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

                If Not colorValue.HasMultipleValues Then
                    oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
                Else
                    oWriter.WriteLine(indentChars & "  Colors")

                    Dim colors() As Color
                    colors = colorValue.Values

                    For i = 0 To UBound(colors)
                        oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
                    Next
                End If
            Case kAssetValueTypeFilename
                oWriter.WriteLine(indentChars & "  Type: Filename")

                Dim filenameValue As FilenameAssetValue
                filenameValue = InValue

                oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
            Case kAssetValueTypeFloat
                oWriter.WriteLine(indentChars & "  Type: Float")

                Dim floatValue As FloatAssetValue
                floatValue = InValue

                oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
            Case kAssetValueTypeInteger
                oWriter.WriteLine(indentChars & "  Type: Integer")

                Dim integerValue As IntegerAssetValue
                integerValue = InValue

                oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
            Case kAssetValueTypeReference
                ' This value type is not currently used in any of the assets.
                oWriter.WriteLine(indentChars & "  Type: Reference")

                Dim refType As ReferenceAssetValue
                refType = InValue
            Case kAssetValueTypeString
                oWriter.WriteLine(indentChars & "  Type: String")

                Dim stringValue As StringAssetValue
                stringValue = InValue

                oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
        End Select
    End Sub


    ' Utility function that returns a string with the R,G,B,K values for an input Color object.
    Private Function ColorString(InColor As Color) As String
        ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
    End Function
End Class
```

Write out all document appearances

Description

This sample writes out information about all of the appearances in the active document. This can be useful when trying to use the API to modify existing appearances by allowing you to easily see what information is available for an appearance.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpDocumentAppearances()
    ' Check that a part or assembly document is active.
    If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Ther
        MsgBox "A part or assembly must be active."
        Exit Sub
    End If

    Dim doc As Document
    Set doc = ThisApplication.ActiveDocument

    ' Open a file to write the results.
    Open "C:\Temp\DocumentAppearanceDump.txt" For Output As #1

    Print #1, "Appearances in " & doc.FullFileName

    Dim appearance As Asset
    For Each appearance In doc.AppearanceAssets
        Print #1, "    Appearance"
```

```
        Print #1, "      DisplayName: " & appearance.DisplayName
        Print #1, "      HasTexture: " & appearance.HasTexture
        Print #1, "      IsReadOnly: " & appearance.IsReadOnly
        Print #1, "      Name: " & appearance.Name

        Dim value As AssetValue
        For Each value In appearance
            Call PrintAssetValue(value, 8)
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\DocumentAppearanceDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
            Else
                Print #1, indentChars & "  Colors"

                Dim colors() As color
                colors = colorValue.Values
```

```
                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub

' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
        Dim oWriter As System.IO.StreamWriter
        Sub Main
    ' Check that a part or assembly document is active.
    If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Ther
        MsgBox("A part or assembly must be active.")
        Exit Sub
    End If

    Dim doc As Document
    doc = ThisApplication.ActiveDocument

    ' Open a file to write the results.
    oWriter = New StreamWriter("C:\Temp\DocumentAppearanceDump.txt")

    oWriter.WriteLine("Appearances in " & doc.FullFileName)

    Dim appearance As Asset
    For Each appearance In doc.AppearanceAssets
        oWriter.WriteLine("    Appearance")
        oWriter.WriteLine("      DisplayName: " & appearance.DisplayName)
        oWriter.WriteLine("      HasTexture: " & appearance.HasTexture)
        oWriter.WriteLine("      IsReadOnly: " & appearance.IsReadOnly)
        oWriter.WriteLine("      Name: " & appearance.Name)

        Dim value As AssetValue
        For Each value In appearance
            Call PrintAssetValue(value, 8)
        Next
    Next

    oWriter.Close

    MsgBox("Finished writing output to ""C:\Temp\DocumentAppearanceDump.txt""")
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    oWriter.WriteLine(indentChars & "Value")
    oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
    oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
    oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            oWriter.WriteLine(indentChars & "  Type: Texture")

            Dim textureValue As TextureAssetValue
            textureValue = InValue

            Dim texture As AssetTexture
            texture = textureValue.Value

            Select Case texture.TextureType
```

```
            Case kTextureTypeBitmap
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
            Case kTextureTypeChecker
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
            Case kTextureTypeGradient
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
            Case kTextureTypeMarble
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
            Case kTextureTypeNoise
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
            Case kTextureTypeSpeckle
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
            Case kTextureTypeTile
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
            Case kTextureTypeUnknown
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
            Case kTextureTypeWave
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
            Case kTextureTypeWood
                oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
            Case Else
                oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
        End Select

        oWriter.WriteLine(indentChars & "  Values")
        Dim textureSubValue As AssetValue
        For Each textureSubValue In texture
            Call PrintAssetValue(textureSubValue, Indent + 4)
        Next
    Case kAssetValueTypeBoolean
        oWriter.WriteLine(indentChars & "  Type: Boolean")

        Dim booleanValue As BooleanAssetValue
        booleanValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
    Case kAssetValueTypeChoice
        oWriter.WriteLine(indentChars & "  Type: Choice")

        Dim choiceValue As ChoiceAssetValue
        choiceValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

        Dim names() As String = New String() {}
        Dim choices() As String = New String() {}
        Call choiceValue.GetChoices(names, choices)
        oWriter.WriteLine(indentChars & "    Choices:")
        Dim i As Integer
        For i = 0 To UBound(names)
            oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
        Next
    Case kAssetValueTypeColor
        oWriter.WriteLine(indentChars & "  Type: Color")

        Dim colorValue As ColorAssetValue
        colorValue = InValue

        oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
        oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

        If Not colorValue.HasMultipleValues Then
            oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
        Else
            oWriter.WriteLine(indentChars & "  Colors")

            Dim colors() As Color
            colors = colorValue.Values

            For i = 0 To UBound(colors)
                oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
            Next
        End If
    Case kAssetValueTypeFilename
        oWriter.WriteLine(indentChars & "  Type: Filename")

        Dim filenameValue As FilenameAssetValue
        filenameValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
    Case kAssetValueTypeFloat
        oWriter.WriteLine(indentChars & "  Type: Float")

        Dim floatValue As FloatAssetValue
        floatValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
    Case kAssetValueTypeInteger
        oWriter.WriteLine(indentChars & "  Type: Integer")

        Dim integerValue As IntegerAssetValue
        integerValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
    Case kAssetValueTypeReference
        ' This value type is not currently used in any of the assets.
        oWriter.WriteLine(indentChars & "  Type: Reference")

        Dim refType As ReferenceAssetValue
        refType = InValue
    Case kAssetValueTypeString
        oWriter.WriteLine(indentChars & "  Type: String")

        Dim stringValue As StringAssetValue
        stringValue = InValue

        oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
```

```
    End Select
End Sub

' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As Color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
End Class
```

RemoveAssemblyOverrides

Description

Removes all appearance overrides that have been assigned in the active assembly.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub RemoveAssemblyOverrides()
    ' Get the active assembly document.
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.ActiveDocument

    ' Iterate through the objects that have an override.
    Dim obj As ComponentOccurrence
    For Each obj In asmDoc.ComponentDefinition.AppearanceOverridesObjects
        ' Set it so the occurrence uses the original color of the part.
        obj.AppearanceSourceType = kPartAppearance
    Next
End Sub
```

Copy Code

```
    ' Get the active assembly document.
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.ActiveDocument

    ' Iterate through the objects that have an override.
    Dim obj As ComponentOccurrence
    For Each obj In asmDoc.ComponentDefinition.AppearanceOverridesObjects
        ' Set it so the occurrence uses the original color of the part.
        obj.AppearanceSourceType = kPartAppearance
    Next
```

Removes all appearance overrides in a part.

Description

This sample removes all appearance overrides that have been placed on a part.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub RemovePartOverrides()
    ' Get the active part document.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Iterate through the objects that have an override.
    Dim obj As Object
    For Each obj In partDoc.ComponentDefinition.AppearanceOverridesObjects
        ' Set the source of the appearance based on the type of object.
        ' It's possible to use kPartAppearance in all cases, but this sets
        ' it to the default source used by Inventor when no overrides exist.
        If TypeOf obj Is SurfaceBody Then
            obj.AppearanceSourceType = kPartAppearance
        ElseIf TypeOf obj Is PartFeature Then
            obj.AppearanceSourceType = kBodyAppearance
        ElseIf TypeOf obj Is Face Then
            obj.AppearanceSourceType = kFeatureAppearance
        Else
            MsgBox "Unexpected type with appearance override: " & TypeName(obj)
        End If
    Next
End Sub
```

Copy Code

```
    ' Get the active part document.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    ' Iterate through the objects that have an override.
    Dim obj As Object
    For Each obj In partDoc.ComponentDefinition.AppearanceOverridesObjects
        ' Set the source of the appearance based on the type of object.
        ' It's possible to use kPartAppearance in all cases, but this sets
        ' it to the default source used by Inventor when no overrides exist.
        If TypeOf obj Is SurfaceBody Then
            obj.AppearanceSourceType = kPartAppearance
        ElseIf TypeOf obj Is PartFeature Then
            obj.AppearanceSourceType = kBodyAppearance
        ElseIf TypeOf obj Is Face Then
            obj.AppearanceSourceType = kFeatureAppearance
        Else
            MsgBox("Unexpected type with appearance override: " & TypeName(obj))
        End If
    Next
```

Set the appearance of an occurrence.

Description

Sets the appearance of a selected occurrence in an assembly.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SetOccurrenceAppearance()
    Dim asmDoc As AssemblyDocument
    Set asmDoc = ThisApplication.ActiveDocument

    ' Get an appearance from the document.  To assign an appearance is must
    ' exist in the document.  This looks for a local appearance and if that
    ' fails it copies the appearance from a library to the document.
    Dim localAsset As Asset
    On Error Resume Next
    Set localAsset = asmDoc.Assets.Item("Bamboo")
    If Err Then
        On Error GoTo 0

        ' Failed to get the appearance in the document, so import it.

        ' Get an asset library by name.  Either the displayed name (which
        ' can changed based on the current language) or the internal name
        ' (which is always the same) can be used.
        Dim assetLib As AssetLibrary
        Set assetLib = ThisApplication.AssetLibraries.Item("Autodesk Appearance Library")
        'Set assetLib = ThisApplication.AssetLibraries.Item("314DE259-5443-4621-BFBD-1730C6CC9AE9")

        ' Get an asset in the library.  Again, either the displayed name or the internal
        ' name can be used.
        Dim libAsset As Asset
        Set libAsset = assetLib.AppearanceAssets.Item("Bamboo")
        'Set libAsset = assetLib.AppearanceAssets.Item("ACADGen-082")

        ' Copy the asset locally.
        Set localAsset = libAsset.CopyTo(asmDoc)
    End If
    On Error GoTo 0

    ' Have an occurrence selected.
    Dim occ As ComponentOccurrence
    Set occ = ThisApplication.CommandManager.Pick(kAssemblyOccurrenceFilter, "Select an occurrence.")

    ' Assign the asset to the occurrence.
    occ.appearance = localAsset
End Sub
```

Copy Code

```
    Dim asmDoc As AssemblyDocument
    asmDoc = ThisApplication.ActiveDocument

    ' Get an appearance from the document.  To assign an appearance is must
    ' exist in the document.  This looks for a local appearance and if that
    ' fails it copies the appearance from a library to the document.
    Dim localAsset As Asset
    On Error Resume Next
    localAsset = asmDoc.Assets.Item("Bamboo")
    If Err.Number > 0 Then
        On Error GoTo 0

        ' Failed to get the appearance in the document, so import it.

        ' Get an asset library by name.  Either the displayed name (which
        ' can changed based on the current language) or the internal name
        ' (which is always the same) can be used.
        Dim assetLib As AssetLibrary
        assetLib = ThisApplication.AssetLibraries.Item("Autodesk Appearance Library")
        'assetLib = ThisApplication.AssetLibraries.Item("314DE259-5443-4621-BFBD-1730C6CC9AE9")
```

```
        ' Get an asset in the library.  Again, either the displayed name or the internal
        ' name can be used.
        Dim libAsset As Asset
        libAsset = assetLib.AppearanceAssets.Item("Bamboo")
        'libAsset = assetLib.AppearanceAssets.Item("ACADGen-082")

        ' Copy the asset locally.
        localAsset = libAsset.CopyTo(asmDoc)
    End If
    On Error GoTo 0

    ' Have an occurrence selected.
    Dim occ As ComponentOccurrence
    occ = ThisApplication.CommandManager.Pick(kAssemblyOccurrenceFilter, "Select an occurrence.")

    ' Assign the asset to the occurrence.
    occ.appearance = localAsset
```

Write out all materials to a file.

Description

This sample writes out information about all of the materials in all libraries. This can be useful when trying to use the API to modify existing materials by allowing to easily see what information is available for a material.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpAllMaterialsInAllLibraries()
    ' Open a file to write the results.
    Open "C:\Temp\AllLibMaterialDump.txt" For Output As #1

    ' Iterate through the libraries.
    Dim assetLib As AssetLibrary
    For Each assetLib In ThisApplication.AssetLibraries
        Print #1, "Library" & assetLib.DisplayName
        Print #1, "  DisplayName: " & assetLib.DisplayName
        Print #1, "  FullFileName: " & assetLib.FullFileName
        Print #1, "  InternalName: " & assetLib.InternalName
        Print #1, "  IsReadOnly: " & assetLib.IsReadOnly

        Dim material As MaterialAsset
        For Each material In assetLib.MaterialAssets
            Print #1, "    Material"
            Print #1, "      DisplayName: " & material.DisplayName
            Print #1, "      Category: " & material.CategoryName
            Print #1, "      Associated Appearance: " & material.AppearanceAsset.DisplayName
            Print #1, "      Associated Physical Properties: " & material.PhysicalPropertiesAsset.DisplayName
            Print #1, "      IsReadOnly: " & material.IsReadOnly
            Print #1, "      Name: " & material.Name

            Dim value As AssetValue
            For Each value In material
                Call PrintAssetValue(value, 8)
            Next
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\AllLibMaterialDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
```

```
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
            Else
                Print #1, indentChars & "  Colors"

                Dim colors() As color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub


' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
	Dim oWriter As System.IO.StreamWriter
	Sub Main
		' Open a file to write the results.
		oWriter = New StreamWriter("C:\Temp\AllLibMaterialDump.txt")

		' Iterate through the libraries.
		Dim assetLib As AssetLibrary
		For Each assetLib In ThisApplication.AssetLibraries
			oWriter.WriteLine("Library" & assetLib.DisplayName)
			oWriter.WriteLine("  DisplayName: " & assetLib.DisplayName)
			oWriter.WriteLine("  FullFileName: " & assetLib.FullFileName)
			oWriter.WriteLine("  InternalName: " & assetLib.InternalName)
			oWriter.WriteLine("  IsReadOnly: " & assetLib.IsReadOnly)

			Dim material As MaterialAsset
			For Each material In assetLib.MaterialAssets
				oWriter.WriteLine("    Material")
				oWriter.WriteLine("      DisplayName: " & material.DisplayName)
				oWriter.WriteLine("      Category: " & material.CategoryName)
				oWriter.WriteLine("      Associated Appearance: " & material.AppearanceAsset.DisplayName)
				oWriter.WriteLine("      Associated Physical Properties: " & material.PhysicalPropertiesAsset.DisplayN
				oWriter.WriteLine("      IsReadOnly: " & material.IsReadOnly)
				oWriter.WriteLine("      Name: " & material.Name)

				Dim value As AssetValue
				For Each value In material
					Call PrintAssetValue(value, 8)
				Next
			Next
		Next

		oWriter.Close()

		MsgBox("Finished writing output to ""C:\Temp\AllLibMaterialDump.txt""")
	End Sub


	' Utility function that prints out information for the input asset value.
	Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
		Dim indentChars As String
		indentChars = Space(Indent)

		oWriter.WriteLine(indentChars & "Value")
		oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
		oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
		oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

		Select Case InValue.ValueType
			Case kAssetValueTextureType
				oWriter.WriteLine(indentChars & "  Type: Texture")

				Dim textureValue As TextureAssetValue
				textureValue = InValue

				Dim texture As AssetTexture
				texture = textureValue.Value

				Select Case texture.TextureType
					Case kTextureTypeBitmap
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
					Case kTextureTypeChecker
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
					Case kTextureTypeGradient
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
					Case kTextureTypeMarble
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
					Case kTextureTypeNoise
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
					Case kTextureTypeSpeckle
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
					Case kTextureTypeTile
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
					Case kTextureTypeUnknown
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
					Case kTextureTypeWave
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
					Case kTextureTypeWood
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
					Case Else
						oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
				End Select

				oWriter.WriteLine(indentChars & "  Values")
				Dim textureSubValue As AssetValue
				For Each textureSubValue In texture
					Call PrintAssetValue(textureSubValue, Indent + 4)
				Next
			Case kAssetValueTypeBoolean
				oWriter.WriteLine(indentChars & "  Type: Boolean")

				Dim booleanValue As BooleanAssetValue
				booleanValue = InValue

				oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
			Case kAssetValueTypeChoice
				oWriter.WriteLine(indentChars & "  Type: Choice")

				Dim choiceValue As ChoiceAssetValue
				choiceValue = InValue

				oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

				Dim names() As String = New String() {}
```

```
                        Dim choices() As String = New String() {}
                        Call choiceValue.GetChoices(names, choices)
                        oWriter.WriteLine(indentChars & "    Choices:")
                        Dim i As Integer
                        For i = 0 To UBound(names)
                                oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
                        Next
                Case kAssetValueTypeColor
                        oWriter.WriteLine(indentChars & "  Type: Color")

                        Dim colorValue As ColorAssetValue
                        colorValue = InValue

                        oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
                        oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

                        If Not colorValue.HasMultipleValues Then
                                oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
                        Else
                                oWriter.WriteLine(indentChars & "  Colors")

                                Dim colors() As Color
                                colors = colorValue.Values

                                For i = 0 To UBound(colors)
                                        oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
                                Next
                        End If
                Case kAssetValueTypeFilename
                        oWriter.WriteLine(indentChars & "  Type: Filename")

                        Dim filenameValue As FilenameAssetValue
                        filenameValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
                Case kAssetValueTypeFloat
                        oWriter.WriteLine(indentChars & "  Type: Float")

                        Dim floatValue As FloatAssetValue
                        floatValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
                Case kAssetValueTypeInteger
                        oWriter.WriteLine(indentChars & "  Type: Integer")

                        Dim integerValue As IntegerAssetValue
                        integerValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
                Case kAssetValueTypeReference
                        ' This value type is not currently used in any of the assets.
                        oWriter.WriteLine(indentChars & "  Type: Reference")

                        Dim refType As ReferenceAssetValue
                        refType = InValue
                Case kAssetValueTypeString
                        oWriter.WriteLine(indentChars & "  Type: String")

                        Dim stringValue As StringAssetValue
                        stringValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
        End Select
    End Sub


    ' Utility function that returns a string with the R,G,B,K values for an input Color object.
    Private Function ColorString(InColor As Color) As String
        ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
    End Function
End Class
```

Write out all physical properties to a file.

Description

This sample writes out information about all of the physical properties in all libraries. This can be useful when trying to use the API to modify existing materials by allowing to easily see what information is available for a physical property.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpAllPhysicalPropertiesInAllLibraries()
    ' Open a file to write the results.
    Open "C:\Temp\AllLibPhysicalPropertiesDump.txt" For Output As #1

    ' Iterate through the libraries.
    Dim assetLib As AssetLibrary
    For Each assetLib In ThisApplication.AssetLibraries
        Print #1, "Library" & assetLib.DisplayName
        Print #1, "  DisplayName: " & assetLib.DisplayName
        Print #1, "  FullFileName: " & assetLib.FullFileName
        Print #1, "  InternalName: " & assetLib.InternalName
        Print #1, "  IsReadOnly: " & assetLib.IsReadOnly
```

```
        Dim physicalAsset As Asset
        For Each physicalAsset In assetLib.PhysicalAssets
            Print #1, "    Physical Property"
            Print #1, "      DisplayName: " & physicalAsset.DisplayName
            Print #1, "      IsReadOnly: " & physicalAsset.IsReadOnly
            Print #1, "      Name: " & physicalAsset.Name

            Dim value As AssetValue
            For Each value In physicalAsset
                Call PrintAssetValue(value, 8)
            Next
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\AllLibPhysicalPropertiesDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
            Else
                Print #1, indentChars & "  Colors"
```

```
                Dim colors() As color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub


' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
        Dim oWriter As System.IO.StreamWriter
        Sub Main
                ' Open a file to write the results.
                oWriter = New StreamWriter("C:\Temp\AllLibPhysicalPropertiesDump.txt")

                ' Iterate through the libraries.
                Dim assetLib As AssetLibrary
                For Each assetLib In ThisApplication.AssetLibraries
                        oWriter.WriteLine("Library" & assetLib.DisplayName)
                        oWriter.WriteLine("  DisplayName: " & assetLib.DisplayName)
                        oWriter.WriteLine("  FullFileName: " & assetLib.FullFileName)
                        oWriter.WriteLine("  InternalName: " & assetLib.InternalName)
                        oWriter.WriteLine("  IsReadOnly: " & assetLib.IsReadOnly)

                        Dim physicalAsset As Asset
                        For Each physicalAsset In assetLib.PhysicalAssets
                                oWriter.WriteLine("    Physical Property")
                                oWriter.WriteLine("      DisplayName: " & physicalAsset.DisplayName)
                                oWriter.WriteLine("      IsReadOnly: " & physicalAsset.IsReadOnly)
                                oWriter.WriteLine("      Name: " & physicalAsset.Name)

                                Dim value As AssetValue
                                For Each value In physicalAsset
                                        Call PrintAssetValue(value, 8)
                                Next
                        Next
                Next

                oWriter.Close

                MsgBox("Finished writing output to ""C:\Temp\AllLibPhysicalPropertiesDump.txt""")
        End Sub


        ' Utility function that prints out information for the input asset value.
        Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
                Dim indentChars As String
                indentChars = Space(Indent)

                oWriter.WriteLine(indentChars & "Value")
                oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
                oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
                oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

                Select Case InValue.ValueType
                        Case kAssetValueTextureType
                                oWriter.WriteLine(indentChars & "  Type: Texture")

                                Dim textureValue As TextureAssetValue
                                textureValue = InValue

                                Dim texture As AssetTexture
```

```
        texture = textureValue.Value

        Select Case texture.TextureType
                Case kTextureTypeBitmap
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
                Case kTextureTypeChecker
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
                Case kTextureTypeGradient
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
                Case kTextureTypeMarble
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
                Case kTextureTypeNoise
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
                Case kTextureTypeSpeckle
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
                Case kTextureTypeTile
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
                Case kTextureTypeUnknown
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
                Case kTextureTypeWave
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
                Case kTextureTypeWood
                        oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
                Case Else
                        oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
        End Select

        oWriter.WriteLine(indentChars & "  Values")
        Dim textureSubValue As AssetValue
        For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
        Next
Case kAssetValueTypeBoolean
        oWriter.WriteLine(indentChars & "  Type: Boolean")

        Dim booleanValue As BooleanAssetValue
        booleanValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
Case kAssetValueTypeChoice
        oWriter.WriteLine(indentChars & "  Type: Choice")

        Dim choiceValue As ChoiceAssetValue
        choiceValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

        Dim names() As String = New String() {}
        Dim choices() As String = New String() {}
        Call choiceValue.GetChoices(names, choices)
        oWriter.WriteLine(indentChars & "    Choices:")
        Dim i As Integer
        For i = 0 To UBound(names)
                oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
        Next
Case kAssetValueTypeColor
        oWriter.WriteLine(indentChars & "  Type: Color")

        Dim colorValue As ColorAssetValue
        colorValue = InValue

        oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
        oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

        If Not colorValue.HasMultipleValues Then
                oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
        Else
                oWriter.WriteLine(indentChars & "  Colors")

                Dim colors() As Color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                        oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
                Next
        End If
Case kAssetValueTypeFilename
        oWriter.WriteLine(indentChars & "  Type: Filename")

        Dim filenameValue As FilenameAssetValue
        filenameValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
Case kAssetValueTypeFloat
        oWriter.WriteLine(indentChars & "  Type: Float")

        Dim floatValue As FloatAssetValue
        floatValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
Case kAssetValueTypeInteger
        oWriter.WriteLine(indentChars & "  Type: Integer")

        Dim integerValue As IntegerAssetValue
        integerValue = InValue

        oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
Case kAssetValueTypeReference
        ' This value type is not currently used in any of the assets.
        oWriter.WriteLine(indentChars & "  Type: Reference")

        Dim refType As ReferenceAssetValue
        refType = InValue
Case kAssetValueTypeString
        oWriter.WriteLine(indentChars & "  Type: String")

        Dim stringValue As StringAssetValue
```

```
                                stringValue = InValue

                                oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
                End Select
        End Sub


        ' Utility function that returns a string with the R,G,B,K values for an input Color object.
        Private Function ColorString(InColor As Color) As String
                ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
        End Function
End Class
```

Write out all document materials to a file.

Description

This sample writes out information about all of the materials in the active document. This can be useful when trying to use the API to modify existing materials by allowing you to easily see what information is available for a material.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpDocumentMaterials()
    ' Check that a part or assembly document is active.
    If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Ther
        MsgBox "A part or assembly must be active."
        Exit Sub
    End If

    Dim doc As Document
    Set doc = ThisApplication.ActiveDocument

    ' Open a file to write the results.
    Open "C:\Temp\DocumentMaterialDump.txt" For Output As #1

    Print #1, "Materials in " & doc.FullFileName

    ' Iterate through the libraries.
    Dim material As MaterialAsset
    For Each material In doc.MaterialAssets
        Print #1, "    Material"
        Print #1, "      DisplayName: " & material.DisplayName
        Print #1, "      Category: " & material.CategoryName
        Print #1, "      Associated Appearance: " & material.AppearanceAsset.DisplayName
        Print #1, "      Associated Physical Properties: " & material.PhysicalPropertiesAsset.DisplayName
        Print #1, "      IsReadOnly: " & material.IsReadOnly
        Print #1, "      Name: " & material.Name

        Dim value As AssetValue
        For Each value In material
            Call PrintAssetValue(value, 8)
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\DocumentMaterialDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
```

```
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
            Else
                Print #1, indentChars & "  Colors"

                Dim colors() As color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub


' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
	Dim oWriter As System.IO.StreamWriter
	Sub Main
		' Check that a part or assembly document is active.
		If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocume
			MsgBox("A part or assembly must be active.")
			Exit Sub
		End If

		Dim doc As Document
		doc = ThisApplication.ActiveDocument

		' Open a file to write the results.
		oWriter = New StreamWriter("C:\Temp\DocumentMaterialDump.txt")

		oWriter.WriteLine("Materials in " & doc.FullFileName)

		' Iterate through the libraries.
		Dim material As MaterialAsset
		For Each material In doc.MaterialAssets
			oWriter.WriteLine("    Material")
			oWriter.WriteLine("      DisplayName: " & material.DisplayName)
			oWriter.WriteLine("      Category: " & material.CategoryName)
			oWriter.WriteLine("      Associated Appearance: " & material.AppearanceAsset.DisplayName)
			oWriter.WriteLine("      Associated Physical Properties: " & material.PhysicalPropertiesAsset.DisplayName)
			oWriter.WriteLine("      IsReadOnly: " & material.IsReadOnly)
			oWriter.WriteLine("      Name: " & material.Name)

			Dim value As AssetValue
			For Each value In material
				Call PrintAssetValue(value, 8)
			Next
		Next

		oWriter.Close()

		MsgBox("Finished writing output to ""C:\Temp\DocumentMaterialDump.txt""")
	End Sub


	' Utility function that prints out information for the input asset value.
	Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
		Dim indentChars As String
		indentChars = Space(Indent)

		oWriter.WriteLine(indentChars & "Value")
		oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
		oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
		oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

		Select Case InValue.ValueType
			Case kAssetValueTextureType
				oWriter.WriteLine(indentChars & "  Type: Texture")

				Dim textureValue As TextureAssetValue
				textureValue = InValue

				Dim texture As AssetTexture
				texture = textureValue.Value

				Select Case texture.TextureType
					Case kTextureTypeBitmap
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
					Case kTextureTypeChecker
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
					Case kTextureTypeGradient
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
					Case kTextureTypeMarble
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
					Case kTextureTypeNoise
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
					Case kTextureTypeSpeckle
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
					Case kTextureTypeTile
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
					Case kTextureTypeUnknown
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
					Case kTextureTypeWave
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
					Case kTextureTypeWood
						oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
					Case Else
						oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
				End Select

				oWriter.WriteLine(indentChars & "  Values")
				Dim textureSubValue As AssetValue
				For Each textureSubValue In texture
					Call PrintAssetValue(textureSubValue, Indent + 4)
				Next
			Case kAssetValueTypeBoolean
				oWriter.WriteLine(indentChars & "  Type: Boolean")

				Dim booleanValue As BooleanAssetValue
				booleanValue = InValue

				oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
			Case kAssetValueTypeChoice
				oWriter.WriteLine(indentChars & "  Type: Choice")

				Dim choiceValue As ChoiceAssetValue
				choiceValue = InValue

				oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

				Dim names() As String = New String() {}
```

```
                        Dim choices() As String = New String() {}
                        Call choiceValue.GetChoices(names, choices)
                        oWriter.WriteLine(indentChars & "    Choices:")
                        Dim i As Integer
                        For i = 0 To UBound(names)
                                oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
                        Next
                Case kAssetValueTypeColor
                        oWriter.WriteLine(indentChars & "  Type: Color")

                        Dim colorValue As ColorAssetValue
                        colorValue = InValue

                        oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
                        oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

                        If Not colorValue.HasMultipleValues Then
                                oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
                        Else
                                oWriter.WriteLine(indentChars & "  Colors")

                                Dim colors() As Color
                                colors = colorValue.Values

                                For i = 0 To UBound(colors)
                                        oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
                                Next
                        End If
                Case kAssetValueTypeFilename
                        oWriter.WriteLine(indentChars & "  Type: Filename")

                        Dim filenameValue As FilenameAssetValue
                        filenameValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
                Case kAssetValueTypeFloat
                        oWriter.WriteLine(indentChars & "  Type: Float")

                        Dim floatValue As FloatAssetValue
                        floatValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
                Case kAssetValueTypeInteger
                        oWriter.WriteLine(indentChars & "  Type: Integer")

                        Dim integerValue As IntegerAssetValue
                        integerValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
                Case kAssetValueTypeReference
                        ' This value type is not currently used in any of the assets.
                        oWriter.WriteLine(indentChars & "  Type: Reference")

                        Dim refType As ReferenceAssetValue
                        refType = InValue
                Case kAssetValueTypeString
                        oWriter.WriteLine(indentChars & "  Type: String")

                        Dim stringValue As StringAssetValue
                        stringValue = InValue

                        oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
        End Select
    End Sub


    ' Utility function that returns a string with the R,G,B,K values for an input Color object.
    Private Function ColorString(InColor As Color) As String
        ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
    End Function
End Class
```

Write out all document physical properties to a file.

Description

This sample writes out information about all of the physical properties in the active document. This can be useful when trying to use the API to modify existing materials by allowing you to easily see what information is available for a physical property

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DumpDocumentPhysicalProperties()
    ' Check that a part or assembly document is active.
    If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocumentObject Then
        MsgBox "A part or assembly must be active."
        Exit Sub
    End If

    Dim doc As Document
    Set doc = ThisApplication.ActiveDocument

    ' Open a file to write the results.
    Open "C:\Temp\DocumentPhysicalPropertiesDump.txt" For Output As #1
```

```
    Print #1, "Physical properties in " & doc.FullFileName

    ' Iterate through the libraries.
    Dim physicalAsset As Asset
    For Each physicalAsset In doc.PhysicalAssets
        Print #1, "    Physical Asset"
        Print #1, "      DisplayName: " & physicalAsset.DisplayName
        Print #1, "      IsReadOnly: " & physicalAsset.IsReadOnly
        Print #1, "      Name: " & physicalAsset.Name

        Dim value As AssetValue
        For Each value In physicalAsset
            Call PrintAssetValue(value, 8)
        Next
    Next

    Close #1

    MsgBox "Finished writing output to ""C:\Temp\DocumentPhysicalPropertiesDump.txt"""
End Sub


' Utility function that prints out information for the input asset value.
Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
    Dim indentChars As String
    indentChars = Space(Indent)

    Print #1, indentChars & "Value"
    Print #1, indentChars & "  DisplayName: " & InValue.DisplayName
    Print #1, indentChars & "  Name: " & InValue.Name
    Print #1, indentChars & "  IsReadOnly: " & InValue.IsReadOnly

    Select Case InValue.ValueType
        Case kAssetValueTextureType
            Print #1, indentChars & "  Type: Texture"

            Dim textureValue As TextureAssetValue
            Set textureValue = InValue

            Dim texture As AssetTexture
            Set texture = textureValue.value

            Select Case texture.TextureType
                Case kTextureTypeBitmap
                    Print #1, indentChars & "  TextureType: kTextureTypeBitmap"
                Case kTextureTypeChecker
                    Print #1, indentChars & "  TextureType: kTextureTypeChecker"
                Case kTextureTypeGradient
                    Print #1, indentChars & "  TextureType: kTextureTypeGradient"
                Case kTextureTypeMarble
                    Print #1, indentChars & "  TextureType: kTextureTypeMarble"
                Case kTextureTypeNoise
                    Print #1, indentChars & "  TextureType: kTextureTypeNoise"
                Case kTextureTypeSpeckle
                    Print #1, indentChars & "  TextureType: kTextureTypeSpeckle"
                Case kTextureTypeTile
                    Print #1, indentChars & "  TextureType: kTextureTypeTile"
                Case kTextureTypeUnknown
                    Print #1, indentChars & "  TextureType: kTextureTypeUnknown"
                Case kTextureTypeWave
                    Print #1, indentChars & "  TextureType: kTextureTypeWave"
                Case kTextureTypeWood
                    Print #1, indentChars & "  TextureType: kTextureTypeWood"
                Case Else
                    Print #1, indentChars & "  TextureType: Unexpected type returned"
            End Select

            Print #1, indentChars & "  Values"
            Dim textureSubValue As AssetValue
            For Each textureSubValue In texture
                Call PrintAssetValue(textureSubValue, Indent + 4)
            Next
        Case kAssetValueTypeBoolean
            Print #1, indentChars & "  Type: Boolean"

            Dim booleanValue As BooleanAssetValue
            Set booleanValue = InValue

            Print #1, indentChars & "    Value: " & booleanValue.value
        Case kAssetValueTypeChoice
            Print #1, indentChars & "  Type: Choice"

            Dim choiceValue As ChoiceAssetValue
            Set choiceValue = InValue

            Print #1, indentChars & "    Value: " & choiceValue.value

            Dim names() As String
            Dim choices() As String
            Call choiceValue.GetChoices(names, choices)
            Print #1, indentChars & "    Choices:"
            Dim i As Integer
            For i = 0 To UBound(names)
                Print #1, indentChars & "      " & names(i) & ", " & choices(i)
            Next
        Case kAssetValueTypeColor
            Print #1, indentChars & "  Type: Color"

            Dim colorValue As ColorAssetValue
            Set colorValue = InValue

            Print #1, indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture
            Print #1, indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues

            If Not colorValue.HasMultipleValues Then
                Print #1, indentChars & "  Color: " & ColorString(colorValue.value)
```

```
            Else
                Print #1, indentChars & "  Colors"

                Dim colors() As color
                colors = colorValue.Values

                For i = 0 To UBound(colors)
                    Print #1, indentChars & "    Color: " & ColorString(colors(i))
                Next
            End If
        Case kAssetValueTypeFilename
            Print #1, indentChars & "  Type: Filename"

            Dim filenameValue As FilenameAssetValue
            Set filenameValue = InValue

            Print #1, indentChars & "    Value: " & filenameValue.value
        Case kAssetValueTypeFloat
            Print #1, indentChars & "  Type: Float"

            Dim floatValue As FloatAssetValue
            Set floatValue = InValue

            Print #1, indentChars & "    Value: " & floatValue.value
        Case kAssetValueTypeInteger
            Print #1, indentChars & "  Type: Integer"

            Dim integerValue As IntegerAssetValue
            Set integerValue = InValue

            Print #1, indentChars & "    Value: " & integerValue.value
        Case kAssetValueTypeReference
            ' This value type is not currently used in any of the assets.
            Print #1, indentChars & "  Type: Reference"

            Dim refType As ReferenceAssetValue
            Set refType = InValue
        Case kAssetValueTypeString
            Print #1, indentChars & "  Type: String"

            Dim stringValue As StringAssetValue
            Set stringValue = InValue

            Print #1, indentChars & "    Value: """ & stringValue.value & """"
    End Select
End Sub


' Utility function that returns a string with the R,G,B,K values for an input Color object.
Private Function ColorString(InColor As color) As String
    ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
End Function
```

Copy Code

```
Imports System.IO
Class Test
        Dim oWriter As System.IO.StreamWriter
        Sub Main
                ' Check that a part or assembly document is active.
                If ThisApplication.ActiveDocumentType <> kAssemblyDocumentObject And ThisApplication.ActiveDocumentType <> kPartDocume
                        MsgBox("A part or assembly must be active.")
                        Exit Sub
                End If

                Dim doc As Document
                doc = ThisApplication.ActiveDocument

                ' Open a file to write the results.
                oWriter = New StreamWriter("C:\Temp\DocumentPhysicalPropertiesDump.txt")

                oWriter.WriteLine("Physical properties in " & doc.FullFileName)

                ' Iterate through the libraries.
                Dim physicalAsset As Asset
                For Each physicalAsset In doc.PhysicalAssets
                        oWriter.WriteLine("    Physical Asset")
                        oWriter.WriteLine("      DisplayName: " & physicalAsset.DisplayName)
                        oWriter.WriteLine("      IsReadOnly: " & physicalAsset.IsReadOnly)
                        oWriter.WriteLine("      Name: " & physicalAsset.Name)

                        Dim value As AssetValue
                        For Each value In physicalAsset
                                Call PrintAssetValue(value, 8)
                        Next
                Next

                oWriter.Close()

                MsgBox("Finished writing output to ""C:\Temp\DocumentPhysicalPropertiesDump.txt""")
        End Sub


        ' Utility function that prints out information for the input asset value.
        Private Sub PrintAssetValue(InValue As AssetValue, Indent As Integer)
                Dim indentChars As String
                indentChars = Space(Indent)

                oWriter.WriteLine(indentChars & "Value")
                oWriter.WriteLine(indentChars & "  DisplayName: " & InValue.DisplayName)
                oWriter.WriteLine(indentChars & "  Name: " & InValue.Name)
                oWriter.WriteLine(indentChars & "  IsReadOnly: " & InValue.IsReadOnly)

                Select Case InValue.ValueType
                        Case kAssetValueTextureType
                                oWriter.WriteLine(indentChars & "  Type: Texture")
```

```
		Dim textureValue As TextureAssetValue
		textureValue = InValue

		Dim texture As AssetTexture
		texture = textureValue.Value

		Select Case texture.TextureType
			Case kTextureTypeBitmap
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeBitmap")
			Case kTextureTypeChecker
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeChecker")
			Case kTextureTypeGradient
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeGradient")
			Case kTextureTypeMarble
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeMarble")
			Case kTextureTypeNoise
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeNoise")
			Case kTextureTypeSpeckle
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeSpeckle")
			Case kTextureTypeTile
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeTile")
			Case kTextureTypeUnknown
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeUnknown")
			Case kTextureTypeWave
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWave")
			Case kTextureTypeWood
				oWriter.WriteLine(indentChars & "  TextureType: kTextureTypeWood")
			Case Else
				oWriter.WriteLine(indentChars & "  TextureType: Unexpected type returned")
		End Select

		oWriter.WriteLine(indentChars & "  Values")
		Dim textureSubValue As AssetValue
		For Each textureSubValue In texture
			Call PrintAssetValue(textureSubValue, Indent + 4)
		Next
	Case kAssetValueTypeBoolean
		oWriter.WriteLine(indentChars & "  Type: Boolean")

		Dim booleanValue As BooleanAssetValue
		booleanValue = InValue

		oWriter.WriteLine(indentChars & "    Value: " & booleanValue.Value)
	Case kAssetValueTypeChoice
		oWriter.WriteLine(indentChars & "  Type: Choice")

		Dim choiceValue As ChoiceAssetValue
		choiceValue = InValue

		oWriter.WriteLine(indentChars & "    Value: " & choiceValue.Value)

		Dim names() As String = New String() {}
		Dim choices() As String = New String() {}
		Call choiceValue.GetChoices(names, choices)
		oWriter.WriteLine(indentChars & "    Choices:")
		Dim i As Integer
		For i = 0 To UBound(names)
			oWriter.WriteLine(indentChars & "      " & names(i) & ", " & choices(i))
		Next
	Case kAssetValueTypeColor
		oWriter.WriteLine(indentChars & "  Type: Color")

		Dim colorValue As ColorAssetValue
		colorValue = InValue

		oWriter.WriteLine(indentChars & "  HasConnectedTexture: " & colorValue.HasConnectedTexture)
		oWriter.WriteLine(indentChars & "  HasMultipleValues: " & colorValue.HasMultipleValues)

		If Not colorValue.HasMultipleValues Then
			oWriter.WriteLine(indentChars & "  Color: " & ColorString(colorValue.Value))
		Else
			oWriter.WriteLine(indentChars & "  Colors")

			Dim colors() As Color
			colors = colorValue.Values

			For i = 0 To UBound(colors)
				oWriter.WriteLine(indentChars & "    Color: " & ColorString(colors(i)))
			Next
		End If
	Case kAssetValueTypeFilename
		oWriter.WriteLine(indentChars & "  Type: Filename")

		Dim filenameValue As FilenameAssetValue
		filenameValue = InValue

		oWriter.WriteLine(indentChars & "    Value: " & filenameValue.Value)
	Case kAssetValueTypeFloat
		oWriter.WriteLine(indentChars & "  Type: Float")

		Dim floatValue As FloatAssetValue
		floatValue = InValue

		oWriter.WriteLine(indentChars & "    Value: " & floatValue.Value)
	Case kAssetValueTypeInteger
		oWriter.WriteLine(indentChars & "  Type: Integer")

		Dim integerValue As IntegerAssetValue
		integerValue = InValue

		oWriter.WriteLine(indentChars & "    Value: " & integerValue.Value)
	Case kAssetValueTypeReference
		' This value type is not currently used in any of the assets.
		oWriter.WriteLine(indentChars & "  Type: Reference")

		Dim refType As ReferenceAssetValue
		refType = InValue
```

```
                        Case kAssetValueTypeString
                                oWriter.WriteLine(indentChars & "  Type: String")

                                Dim stringValue As StringAssetValue
                                stringValue = InValue

                                oWriter.WriteLine(indentChars & "    Value: """ & stringValue.Value & """")
                End Select
        End Sub


        ' Utility function that returns a string with the R,G,B,K values for an input Color object.
        Private Function ColorString(InColor As Color) As String
                ColorString = InColor.Red & "," & InColor.Green & "," & InColor.Blue & "," & InColor.Opacity
        End Function
End Class
```

Create a Draft Analysis

Description

This sample demonstrates the creation of a draft analysis in a part.

Code Samples

- VBA
- iLogic

Open a part document and run the following sample.

Copy Code

```
Public Sub CreateDraftAnalysis()
    ' Set a reference to the active part document
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    ' Set a reference to the analysis manager
    Dim oAnalysisMgr As AnalysisManager
    Set oAnalysisMgr = oCompDef.AnalysisManager

    ' Get the y-axis, to be used as the pull direction for the draft analysis
    Dim oYAxis As WorkAxis
    Set oYAxis = oCompDef.WorkAxes.Item(2)

    ' Create the draft analysis with the following input values
    '   Start Angle = -3 degrees
    '   End Angle =   3 degrees
    '   Pull direction = Negative Y Axis
    '   Display Quality = 50%

    Dim oDraftAnalysis As DraftAnalysis
    Set oDraftAnalysis = oAnalysisMgr.DraftAnalyses.Add(-0.05236, 0.05236, oYAxis, True, , , 5)
End Sub
```

Open a part document and run the following sample.

Copy Code

```
    ' Set a reference to the active part document
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition
    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    ' Set a reference to the analysis manager
    Dim oAnalysisMgr As AnalysisManager
    oAnalysisMgr = oCompDef.AnalysisManager

    ' Get the y-axis, to be used as the pull direction for the draft analysis
    Dim oYAxis As WorkAxis
    oYAxis = oCompDef.WorkAxes.Item(2)

    ' Create the draft analysis with the following input values
    '   Start Angle = -3 degrees
    '   End Angle =   3 degrees
    '   Pull direction = Negative Y Axis
    '   Display Quality = 50%

    Dim oDraftAnalysis As DraftAnalysis
    oDraftAnalysis = oAnalysisMgr.DraftAnalyses.Add(-0.05236, 0.05236, oYAxis, True, , , 5)
```

Control point, equation, and intersection curve creation.

Description

This sample demonstrates several new curve creation techniques introduced in Inventor 2014. It creates a new part and then create a 2d control point spline and a 2d equation curve. Surfaces are created from these two curves by extruding them. A 3d intersection curve is created between the extrusions. A 3d control point spline and 3d equation curve are also created.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SketchCurves()
    ' Create a new part.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                  ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a 2D sketch on the X-Y plane.
    Dim sketch1 As PlanarSketch
    Set sketch1 = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Create a spline based on control points.
    Dim pnts As ObjectCollection
    Set pnts = ThisApplication.TransientObjects.CreateObjectCollection
    Call pnts.Add(tg.CreatePoint2d(2, 0))
    Call pnts.Add(tg.CreatePoint2d(4, 1))
    Call pnts.Add(tg.CreatePoint2d(4, 2))
    Call pnts.Add(tg.CreatePoint2d(6, 3))
    Call pnts.Add(tg.CreatePoint2d(8, 1))
    Dim controlPointSpline As SketchControlPointSpline
    Set controlPointSpline = sketch1.SketchControlPointSplines.Add(pnts)

    ' Create a 2D sketch on the Y-Z plane.
    Dim sketch2 As PlanarSketch
    Set sketch2 = partDef.Sketches.Add(partDef.WorkPlanes.Item(1))

    ' Create a spline based on an equation.
    Dim equationCurve As SketchEquationCurve
    Set equationCurve = sketch2.SketchEquationCurves.Add(kParametric, kCartesian, _
                                ".001*t * cos(t)", ".001*t * sin(t)", 0, 360 * 3)

    ' Create a 3D sketch.
    Dim sketch3 As sketch3D
    Set sketch3 = partDef.Sketches3D.Add

    ' Create a 3D spline based on control points.
    Set pnts = ThisApplication.TransientObjects.CreateObjectCollection
    Call pnts.Add(tg.CreatePoint(10, 0, 0))
    Call pnts.Add(tg.CreatePoint(12, 1, 3))
    Call pnts.Add(tg.CreatePoint(12, 2, -5))
    Call pnts.Add(tg.CreatePoint(14, 3, 2))
    Call pnts.Add(tg.CreatePoint(16, 1, -3))
    Dim controlPointSpline2 As SketchControlPointSpline3D
    Set controlPointSpline2 = sketch3.SketchControlPointSplines3D.Add(pnts)

    ' Create a 3D spline based on an equation.
    Dim equationCurve2 As SketchEquationCurve3D
    Set equationCurve2 = sketch3.SketchEquationCurves3D.Add(kCartesian, _
                            ".001*t * cos(t) + 8", ".001*t * sin(t)", "0.002*t", 0, 360 * 3)

    ThisApplication.ActiveView.Fit

    ' Extrude the 2d curves.
    Dim prof As Profile
    Set prof = sketch1.Profiles.AddForSurface(controlPointSpline)
    Dim extrudeDef As ExtrudeDefinition
    Set extrudeDef = partDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(prof, kSurfaceOperation)
    Call extrudeDef.SetDistanceExtent(6, kSymmetricExtentDirection)
    Dim extrude1 As ExtrudeFeature
    Set extrude1 = partDef.Features.ExtrudeFeatures.Add(extrudeDef)

    ' Change the work surface to not be transparent.
    Dim surf As WorkSurface
    Set surf = extrude1.SurfaceBodies.Item(1).Parent
    surf.Translucent = False

    Set prof = sketch2.Profiles.AddForSurface(equationCurve)
    Set extrudeDef = partDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(prof, kSurfaceOperation)
    Call extrudeDef.SetDistanceExtent(9, kPositiveExtentDirection)
    Dim extrude2 As ExtrudeFeature
    Set extrude2 = partDef.Features.ExtrudeFeatures.Add(extrudeDef)

    ' Create a new sketch and an intersection curve.
    Dim interSketch As sketch3D
    Set interSketch = partDef.Sketches3D.Add

    Call interSketch.IntersectionCurves.Add(extrude1.SurfaceBodies.Item(1), extrude2.SurfaceBodies.Item(1))
End Sub
```

Copy Code

```
    ' Create a new part.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                  ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a 2D sketch on the X-Y plane.
    Dim sketch1 As PlanarSketch
    sketch1 = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Create a spline based on control points.
    Dim pnts As ObjectCollection
    pnts = ThisApplication.TransientObjects.CreateObjectCollection
    Call pnts.Add(tg.CreatePoint2d(2, 0))
    Call pnts.Add(tg.CreatePoint2d(4, 1))
    Call pnts.Add(tg.CreatePoint2d(4, 2))
    Call pnts.Add(tg.CreatePoint2d(6, 3))
    Call pnts.Add(tg.CreatePoint2d(8, 1))
    Dim controlPointSpline As SketchControlPointSpline
    controlPointSpline = sketch1.SketchControlPointSplines.Add(pnts)

    ' Create a 2D sketch on the Y-Z plane.
    Dim sketch2 As PlanarSketch
    sketch2 = partDef.Sketches.Add(partDef.WorkPlanes.Item(1))

    ' Create a spline based on an equation.
    Dim equationCurve As SketchEquationCurve
    equationCurve = sketch2.SketchEquationCurves.Add(kParametric, kCartesian, _
                               ".001*t * cos(t)", ".001*t * sin(t)", 0, 360 * 3)

    ' Create a 3D sketch.
    Dim sketch3 As sketch3D
    sketch3 = partDef.Sketches3D.Add

    ' Create a 3D spline based on control points.
    pnts = ThisApplication.TransientObjects.CreateObjectCollection
    Call pnts.Add(tg.CreatePoint(10, 0, 0))
    Call pnts.Add(tg.CreatePoint(12, 1, 3))
    Call pnts.Add(tg.CreatePoint(12, 2, -5))
    Call pnts.Add(tg.CreatePoint(14, 3, 2))
    Call pnts.Add(tg.CreatePoint(16, 1, -3))
    Dim controlPointSpline2 As SketchControlPointSpline3D
    controlPointSpline2 = sketch3.SketchControlPointSplines3D.Add(pnts)

    ' Create a 3D spline based on an equation.
    Dim equationCurve2 As SketchEquationCurve3D
    equationCurve2 = sketch3.SketchEquationCurves3D.Add(kCartesian, _
                          ".001*t * cos(t) + 8", ".001*t * sin(t)", "0.002*t", 0, 360 * 3)

    ThisApplication.ActiveView.Fit

    ' Extrude the 2d curves.
    Dim prof As Profile
    prof = sketch1.Profiles.AddForSurface(controlPointSpline)
    Dim extrudeDef As ExtrudeDefinition
    extrudeDef = partDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(prof, kSurfaceOperation)
    Call extrudeDef.SetDistanceExtent(6, kSymmetricExtentDirection)
    Dim extrude1 As ExtrudeFeature
    extrude1 = partDef.Features.ExtrudeFeatures.Add(extrudeDef)

    ' Change the work surface to not be transparent.
    Dim surf As WorkSurface
    surf = extrude1.SurfaceBodies.Item(1).Parent
    surf.Translucent = False

    prof = sketch2.Profiles.AddForSurface(equationCurve)
    extrudeDef = partDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(prof, kSurfaceOperation)
    Call extrudeDef.SetDistanceExtent(9, kPositiveExtentDirection)
    Dim extrude2 As ExtrudeFeature
    extrude2 = partDef.Features.ExtrudeFeatures.Add(extrudeDef)

    ' Create a new sketch and an intersection curve.
    Dim interSketch As sketch3D
    interSketch = partDef.Sketches3D.Add

    Call interSketch.IntersectionCurves.Add(extrude1.SurfaceBodies.Item(1), extrude2.SurfaceBodies.Item(1))
```

Creates an Arc Length Dimension Constraint

Description

Demonstrates creating an arc length dimension constraint.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub ArcLengthDimConstraintSample()
    ' Get the active part document.
    Dim partDoc As PartDocument
```

```
    Set partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a new sketch.
    Dim sketch As PlanarSketch
    Set sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    Dim dPi As Double
    dPi = Atn(1) * 4

    ' Create an arc.
    Dim arc As SketchArc
    Set arc = sketch.SketchArcs.AddByCenterStartSweepAngle(tg.CreatePoint2d(0, 0), 5, dPi / 8, dPi / 4)

    Dim arcConstraint As ArcLengthDimConstraint
    Set arcConstraint = sketch.DimensionConstraints.AddArcLength(arc, tg.CreatePoint2d(10, 9))
End Sub
```

To use this sample a part must be active.

Copy Code

```
    ' Get the active part document.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a new sketch.
    Dim sketch As PlanarSketch
    sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    Dim dPi As Double
    dPi = Math.Atan(1) * 4

    ' Create an arc.
    Dim arc As SketchArc
    arc = sketch.SketchArcs.AddByCenterStartSweepAngle(tg.CreatePoint2d(0, 0), 5, dPi / 8, dPi / 4)

    Dim arcConstraint As ArcLengthDimConstraint
    arcConstraint = sketch.DimensionConstraints.AddArcLength(arc, tg.CreatePoint2d(10, 9))
```

Create Centerpoint Rectangles

Description

Creates a new sketch containing rectangles created using the two new center point rectangle commands.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub SketchCreation()
    ' Get the active part document.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a new sketch.
    Dim sketch As PlanarSketch
    Set sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Draw rectangles by center point.
    Call sketch.SketchLines.AddAsTwoPointCenteredRectangle(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(8, 3))
    Call sketch.SketchLines.AddAsThreePointCenteredRectangle(tg.CreatePoint2d(20, 0), tg.CreatePoint2d(28, 3), tg.CreatePoint2d(24, 9)

    ThisApplication.ActiveView.Fit
End Sub
```

To use this sample a part must be active.

Copy Code

```
    ' Get the active part document.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a new sketch.
    Dim sketch As PlanarSketch
    sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))
```

```
    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Draw rectangles by center point.
    Call sketch.SketchLines.AddAsTwoPointCenteredRectangle(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(8, 3))
    Call sketch.SketchLines.AddAsThreePointCenteredRectangle(tg.CreatePoint2d(20, 0), tg.CreatePoint2d(28, 3), tg.CreatePoint2d(24, 9)

    ThisApplication.ActiveView.Fit
```

Copy a sketch

Description

This sample demonstrates copying the contents of a sketch into another sketch via the API.

Code Samples

- VBA
- iLogic

Before running the sample, have a part document open that contains a sketch with some sketch entities in it.

Copy Code

```
Public Sub CopySketch()
    ' Set a reference to the active document.
    ' This assumes a part document is active.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oDef As PartComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    ' Set a reference to the first sketch in the part.
    Dim oSketchToCopy As PlanarSketch
    Set oSketchToCopy = oDef.Sketches.Item(1)

    ' Select the sketch to copy.
    Call oDoc.SelectSet.Clear
    Call oDoc.SelectSet.Select(oSketchToCopy)

    ' Execute the copy command.
    Dim oCopyControlDef As ControlDefinition
    Set oCopyControlDef = ThisApplication.CommandManager.ControlDefinitions.Item("AppCopyCmd")
    oCopyControlDef.Execute

    ' Create a new sketch on the XY plane.
    Dim oNewSketch As PlanarSketch
    Set oNewSketch = oDef.Sketches.Add(oDef.WorkPlanes.Item(3))

    ' Put the sketch in edit mode.
    oNewSketch.Edit

    ' Execute the paste command.
    Dim oPasteControlDef As ControlDefinition
    Set oPasteControlDef = ThisApplication.CommandManager.ControlDefinitions.Item("AppPasteCmd")
    oPasteControlDef.Execute

    Dim oSketchEnts As ObjectCollection
    Set oSketchEnts = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oSketchEnt As SketchEntity
    For Each oSketchEnt In oNewSketch.SketchEntities
        Call oSketchEnts.Add(oSketchEnt)
    Next

    ' Translate all sketch entities in the new sketch.
    Call oNewSketch.MoveSketchObjects(oSketchEnts, ThisApplication.TransientGeometry.CreateVector2d(1, 0))
End Sub
```

Before running the sample, have a part document open that contains a sketch with some sketch entities in it.

Copy Code

```
    ' Set a reference to the active document.
    ' This assumes a part document is active.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oDef As PartComponentDefinition
    oDef = oDoc.ComponentDefinition

    ' Set a reference to the first sketch in the part.
    Dim oSketchToCopy As PlanarSketch
    oSketchToCopy = oDef.Sketches.Item(1)

    ' Select the sketch to copy.
    Call oDoc.SelectSet.Clear
    Call oDoc.SelectSet.Select(oSketchToCopy)

    ' Execute the copy command.
    Dim oCopyControlDef As ControlDefinition
    oCopyControlDef = ThisApplication.CommandManager.ControlDefinitions.Item("AppCopyCmd")
    oCopyControlDef.Execute

    ' Create a new sketch on the XY plane.
    Dim oNewSketch As PlanarSketch
```

```
    oNewSketch = oDef.Sketches.Add(oDef.WorkPlanes.Item(3))

    ' Put the sketch in edit mode.
    oNewSketch.Edit

    ' Execute the paste command.
    Dim oPasteControlDef As ControlDefinition
    oPasteControlDef = ThisApplication.CommandManager.ControlDefinitions.Item("AppPasteCmd")
    oPasteControlDef.Execute

    Dim oSketchEnts As ObjectCollection
    oSketchEnts = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oSketchEnt As SketchEntity
    For Each oSketchEnt In oNewSketch.SketchEntities
        Call oSketchEnts.Add(oSketchEnt)
    Next

    ' Translate all sketch entities in the new sketch.
    Call oNewSketch.MoveSketchObjects(oSketchEnts, ThisApplication.TransientGeometry.CreateVector2d(1, 0))
```

ImportedDWGComponent Creation

Description

This sample demonstrates how to create an imported DWG component into Inventor part document, and project the DWG entities onto Inventor planar sketch.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub CreateImportedDWGComponentSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oRefComponents As ReferenceComponents
    Set oRefComponents = oCompDef.ReferenceComponents

    ' Create a ImportedComponentDefinition based on an AutoCAD file.
    Dim oImportedCompDef As ImportedComponentDefinition
    Set oImportedCompDef = oRefComponents.ImportedComponents.CreateDefinition("C:\Temp\ACADDWG.dwg")

    Dim oImportedDWGDef As ImportedDWGComponentDefinition

    If oImportedCompDef.Type = kImportedDWGComponentDefinitionObject Then
        Set oImportedDWGDef = oImportedCompDef
    Else
        End
    End If

    Dim oMatrix As Matrix
    Set oMatrix = ThisApplication.TransientGeometry.CreateMatrix
    oMatrix.SetTranslation ThisApplication.TransientGeometry.CreateVector(0, 0, 10)

    oImportedDWGDef.Transformation = oMatrix

    ' Create the ImportedComponent
    Dim oImportedComponent As ImportedComponent
    Set oImportedComponent = oRefComponents.ImportedComponents.Add(oImportedDWGDef)

    Dim oImportedDWGComponent As ImportedDWGComponent

    If oImportedComponent.Type = kImportedDWGComponentObject Then
        Set oImportedDWGComponent = oImportedComponent

        Dim oSk As PlanarSketch
        Set oSk = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

        ' Get the DWGBlockDefinition for model space.
        Dim oDWGModelSpaceDef As DWGBlockDefinition
        Set oDWGModelSpaceDef = oImportedDWGComponent.ModelSpaceDefinition

        On Error Resume Next
        ' Project DWG entities to planar sketch.
        Dim oDWGEntity As DWGEntity
        For Each oDWGEntity In oDWGModelSpaceDef.Entities

            Call oSk.AddByProjectingEntity(oDWGEntity)
        Next
    End If
End Sub
```

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oRefComponents As ReferenceComponents
```

```
    oRefComponents = oCompDef.ReferenceComponents

    ' Create a ImportedComponentDefinition based on an AutoCAD file.
    Dim oImportedCompDef As ImportedComponentDefinition
    oImportedCompDef = oRefComponents.ImportedComponents.CreateDefinition("C:\Temp\ACADDWG.dwg")

    Dim oImportedDWGDef As ImportedDWGComponentDefinition

    If oImportedCompDef.Type = kImportedDWGComponentDefinitionObject Then
        oImportedDWGDef = oImportedCompDef
    End If

    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix
    oMatrix.SetTranslation(ThisApplication.TransientGeometry.CreateVector(0, 0, 10))

    oImportedDWGDef.Transformation = oMatrix

    ' Create the ImportedComponent
    Dim oImportedComponent As ImportedComponent
    oImportedComponent = oRefComponents.ImportedComponents.Add(oImportedDWGDef)

    Dim oImportedDWGComponent As ImportedDWGComponent

    If oImportedComponent.Type = kImportedDWGComponentObject Then
        oImportedDWGComponent = oImportedComponent

        Dim oSk As PlanarSketch
        oSk = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

        ' Get the DWGBlockDefinition for model space.
        Dim oDWGModelSpaceDef As DWGBlockDefinition
        oDWGModelSpaceDef = oImportedDWGComponent.ModelSpaceDefinition

    On Error Resume Next
        ' Project DWG entities to planar sketch.
        Dim oDWGEntity As DWGEntity
        For Each oDWGEntity In oDWGModelSpaceDef.Entities
            Call oSk.AddByProjectingEntity(oDWGEntity)
        Next
    End If
```

Sketch from Face Silhouette

Description

This sample creates a cylindrical solid, creates a new sketch plane and creates some new sketch lines from the actual edges and the apparent (silhouette) edges of the cylinder.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SilhouetteSample()
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch1 As PlanarSketch
    Set oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a circle on the sketch.
    Dim oCircle As SketchCircle
    Set oCircle = oSketch1.SketchCircles.AddByCenterRadius( _
                              oTransGeom.CreatePoint2d(0, 0), 2)

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch1.Profiles.AddForSolid

    ' Create a solid extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(3, kSymmetricExtentDirection)
    Dim oExtrusion As ExtrudeFeature
    Set oExtrusion = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create another sketch on the Y-Z plane.
    Dim oSketch2 As PlanarSketch
    Set oSketch2 = oCompDef.Sketches.Add(oCompDef.WorkPlanes(1))

    ' Get the cylindrical face of the solid.
    Dim oCylinder As Face
    Set oCylinder = oExtrusion.SideFaces.Item(1)
```

```
    ' Create a sketch line using the silhouette of the cylinder.  The proximity
    ' point determines which of the two silhouette edges will be used.
    Dim oSilhouetteCurve As SketchEntity
    Set oSilhouetteCurve = oSketch2.AddBySilhouette(oCylinder, _
                                    oTransGeom.CreatePoint(0, -1, 0))

    ' Create another sketch line from the silhouette on the
    ' other side of the cylinder.
    Set oSilhouetteCurve = oSketch2.AddBySilhouette(oCylinder, _
                                    oTransGeom.CreatePoint(0, 1, 0))

    ' Create sketch lines from the ends of the cylinder.  This takes
    ' advantage of the fact that a cylinder only has two edges.
    Dim oEndLine As SketchEntity
    Set oEndLine = oSketch2.AddByProjectingEntity( _
                            oExtrusion.StartFaces.Item(1).Edges.Item(1))
    Set oEndLine = oSketch2.AddByProjectingEntity( _
                            oExtrusion.EndFaces.Item(1).Edges.Item(1))
End Sub
```

Copy Code

```
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
               ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch1 As PlanarSketch
    oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a circle on the sketch.
    Dim oCircle As SketchCircle
    oCircle = oSketch1.SketchCircles.AddByCenterRadius( _
                               oTransGeom.CreatePoint2d(0, 0), 2)

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch1.Profiles.AddForSolid

    ' Create a solid extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(3, kSymmetricExtentDirection)
    Dim oExtrusion As ExtrudeFeature
    oExtrusion = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create another sketch on the Y-Z plane.
    Dim oSketch2 As PlanarSketch
    oSketch2 = oCompDef.Sketches.Add(oCompDef.WorkPlanes(1))

    ' Get the cylindrical face of the solid.
    Dim oCylinder As Face
    oCylinder = oExtrusion.SideFaces.Item(1)

    ' Create a sketch line using the silhouette of the cylinder.  The proximity
    ' point determines which of the two silhouette edges will be used.
    Dim oSilhouetteCurve As SketchEntity
    oSilhouetteCurve = oSketch2.AddBySilhouette(oCylinder, _
                                    oTransGeom.CreatePoint(0, -1, 0))

    ' Create another sketch line from the silhouette on the
    ' other side of the cylinder.
    oSilhouetteCurve = oSketch2.AddBySilhouette(oCylinder, _
                                    oTransGeom.CreatePoint(0, 1, 0))

    ' Create sketch lines from the ends of the cylinder.  This takes
    ' advantage of the fact that a cylinder only has two edges.
    Dim oEndLine As SketchEntity
    oEndLine = oSketch2.AddByProjectingEntity( _
                            oExtrusion.StartFaces.Item(1).Edges.Item(1))
    oEndLine = oSketch2.AddByProjectingEntity( _
                            oExtrusion.EndFaces.Item(1).Edges.Item(1))
```

Sketch Edit Orientation

Description

This sample demonstrates modifying the orientation of a sketch.

Code Samples

- VBA
- iLogic

To use this sample, have a part document open that contains a box and run the program.

Copy Code

```
Public Sub ChangeSketchPlane()
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model. This sample assumes a simple
    ' model where at least the first face is a plane. (A box is a good
    ' test case.)
    Dim oFace As Face
    Set oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Create a new sketch
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oFace, True)

    ' Draw a circle at the origin of the sketch plane.
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius( _
    ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
    ThisApplication.TransientGeometry.CreatePoint2d(1, 0))

    MsgBox "The sketch will be moved to another face."

    ' Move the sketch to the second face of the model.
    Set oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(2)
    oSketch.PlanarEntity = oFace

    MsgBox "Current Origin Point: " & oSketch.OriginPointGeometry.X & ", " & _
    oSketch.OriginPointGeometry.Y & ", " & oSketch.OriginPointGeometry.Z & _
    Chr(13) & Chr(13) & _
    "The origin of the sketch will now be set to the center point."

    ' Set the origin point to use the center point work point.
    oSketch.OriginPoint = oCompDef.WorkPoints.Item(1)

    MsgBox "New Origin Point: " & oSketch.OriginPointGeometry.X & ", " & _
    oSketch.OriginPointGeometry.Y & ", " & oSketch.OriginPointGeometry.Z & _
    Chr(13) & Chr(13) & _
    "The origin of the sketch will now be set to the center point."

    MsgBox "Current X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z

    MsgBox "The X axis of the sketch will now be redefined."

    ' Set the axis to be one of the edges of the face.
    oSketch.AxisEntity = oFace.Edges.Item(2)

    MsgBox "New X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z

    MsgBox "The direction of the axis will now be reversed."

    ' Reverse the axis direction.
    oSketch.NaturalAxisDirection = False

    MsgBox "New X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z

    MsgBox "The axis will be changed to define the Y instead of the X axis."

    ' Change the axis definition.
    oSketch.AxisIsX = False
End Sub
```

To use this sample, have a part document open that contains a box and run the program.

Copy Code

```
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model. This sample assumes a simple
    ' model where at least the first face is a plane. (A box is a good
    ' test case.)
    Dim oFace As Face
    oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Create a new sketch
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oFace, True)

    ' Draw a circle at the origin of the sketch plane.
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius( _
    ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
    ThisApplication.TransientGeometry.CreatePoint2d(1, 0))

    MsgBox("The sketch will be moved to another face.")

    ' Move the sketch to the second face of the model.
    oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(2)
    oSketch.PlanarEntity = oFace

    MsgBox("Current Origin Point: " & oSketch.OriginPointGeometry.X & ", " & _
    oSketch.OriginPointGeometry.Y & ", " & oSketch.OriginPointGeometry.Z & _
```

```
    Chr(13) & Chr(13) & _
    "The origin of the sketch will now be set to the center point.")

    ' Set the origin point to use the center point work point.
    oSketch.OriginPoint = oCompDef.WorkPoints.Item(1)

    MsgBox("New Origin Point: " & oSketch.OriginPointGeometry.X & ", " & _
    oSketch.OriginPointGeometry.Y & ", " & oSketch.OriginPointGeometry.Z & _
    Chr(13) & Chr(13) & _
    "The origin of the sketch will now be set to the center point.")

    MsgBox("Current X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z)

    MsgBox("The X axis of the sketch will now be redefined.")

    ' Set the axis to be one of the edges of the face.
    oSketch.AxisEntity = oFace.Edges.Item(2)

    MsgBox("New X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z)

    MsgBox("The direction of the axis will now be reversed.")

    ' Reverse the axis direction.
    oSketch.NaturalAxisDirection = False

    MsgBox("New X axis: " & oSketch.AxisEntityGeometry.Direction.X & ", " & _
    oSketch.AxisEntityGeometry.Direction.Y & ", " & _
    oSketch.AxisEntityGeometry.Direction.Z)

    MsgBox("The axis will be changed to define the Y instead of the X axis.")

    ' Change the axis definition.
    oSketch.AxisIsX = False
```

Sketch Share

Description

This sample demonstrates setting a sketch so it is shared.

Code Samples

- VBA
- iLogic

To use this sample, open a part document open that contains at least one extrusion feature and run the sample. The sketch of the first extrusion feature should not already be shared.

Copy Code

```
Public Sub MakeSketchShared()
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first extrusion feature in the part.
    ' This assumes that an extrusion feature exists in the part.
    Dim oExtrudeFeature As ExtrudeFeature
    Set oExtrudeFeature = oCompDef.Features.ExtrudeFeatures.Item(1)

    ' Set a reference to the sketch of the feature.
    Dim oSketch As PlanarSketch
    Set oSketch = oExtrudeFeature.Definition.Profile.Parent

    ' Share the sketch of the feature.
    oSketch.Shared = True
End Sub
```

To use this sample, open a part document open that contains at least one extrusion feature and run the sample. The sketch of the first extrusion feature should not already be shared.

Copy Code

```
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first extrusion feature in the part.
    ' This assumes that an extrusion feature exists in the part.
    Dim oExtrudeFeature As ExtrudeFeature
    oExtrudeFeature = oCompDef.Features.ExtrudeFeatures.Item(1)

    ' Set a reference to the sketch of the feature.
    Dim oSketch As PlanarSketch
    oSketch = oExtrudeFeature.Definition.Profile.Parent

    ' Share the sketch of the feature.
    oSketch.Shared = True
```

Sketch Add

Description

This sample demonstrates the creation of a sketch using the Sketches.Add method.

Code Samples

- VBA
- iLogic

Before using this sample, open a part document that contains a box.

Copy Code

```
Public Sub AddSketch()
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model.  This sample assumes a simple
    ' model where at least the first face is a plane.  (A box is a good
    ' test case.)
    Dim oFace As Face
    Set oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Create a new sketch.  The second argument specifies to include
    ' the edges of the face in the sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oFace, True)

    ' Change the name.
    oSketch.Name = "My New Sketch"
End Sub
```

Before using this sample, open a part document that contains a box.

Copy Code

```
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model.  This sample assumes a simple
    ' model where at least the first face is a plane.  (A box is a good
    ' test case.)
    Dim oFace As Face
    oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Create a new sketch.  The second argument specifies to include
    ' the edges of the face in the sketch.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oFace, True)

    ' Change the name.
    oSketch.Name = "My New Sketch"
```

Sketch Add Oriented

Description

This sample demonstrates the creation of a sketch using the Sketches.AddWithOrientation method.

Code Samples

- VBA
- iLogic

To use this sample, open a part document that contains a box before running the sample code.

Copy Code

```
Public Sub AddOrientedSketch()
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model.  This sample assumes a simple
    ' model where at least the first face is a plane.  (A box is a good
    ' test case.)
    Dim oFace As Face
    Set oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Get one of the edges of the face to use as the sketch x-axis.
    Dim oEdge As Edge
    Set oEdge = oFace.Edges.Item(2)

    ' Get the start vertex of the edge to use as the origin of the sketch.
    Dim oVertex As Vertex
    Set oVertex = oEdge.StartVertex
```

```
    ' Create a new sketch.  This last argument is set to true to cause the
    ' creation of sketch geometry from the edges of the face.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFace, oEdge, True, _
                                                       True, oVertex, True)

    ' Change the name.
    oSketch.Name = "My Oriented Sketch"

    ' Draw a circle at the origin of the sketch plane.
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
                ThisApplication.TransientGeometry.CreatePoint2d(1, 0))

    ' Create one more sketch on another face with the origin defined by
    ' the center point.  (The first workpoint in the collection is the center
    ' point work point.)  In this case, no sketch geometry will be created
    ' since the final argument has been left out and it defaults to false.
    Set oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(5)
    Set oEdge = oFace.Edges.Item(1)
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFace, oEdge, True, _
                                            True, oCompDef.WorkPoints.Item(1))

    ' Change the name.
    oSketch.Name = "My Origin Sketch"

    ' Draw a circle at the origin of the sketch plane.
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
                ThisApplication.TransientGeometry.CreatePoint2d(1, 0))
End Sub
```

To use this sample, open a part document that contains a box before running the sample code.

Copy Code

```
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get the first face of the model.  This sample assumes a simple
    ' model where at least the first face is a plane.  (A box is a good
    ' test case.)
    Dim oFace As Face
    oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(1)

    ' Get one of the edges of the face to use as the sketch x-axis.
    Dim oEdge As Edge
    oEdge = oFace.Edges.Item(2)

    ' Get the start vertex of the edge to use as the origin of the sketch.
    Dim oVertex As Vertex
    oVertex = oEdge.StartVertex

    ' Create a new sketch.  This last argument is set to true to cause the
    ' creation of sketch geometry from the edges of the face.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.AddWithOrientation(oFace, oEdge, True, _
                                                   True, oVertex, True)

    ' Change the name.
    oSketch.Name = "My Oriented Sketch"

    ' Draw a circle at the origin of the sketch plane.
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
                ThisApplication.TransientGeometry.CreatePoint2d(1, 0))

    ' Create one more sketch on another face with the origin defined by
    ' the center point.  (The first workpoint in the collection is the center
    ' point work point.)  In this case, no sketch geometry will be created
    ' since the final argument has been left out and it defaults to false.
    oFace = oCompDef.SurfaceBodies.Item(1).Faces.Item(5)
    oEdge = oFace.Edges.Item(1)
    oSketch = oCompDef.Sketches.AddWithOrientation(oFace, oEdge, True, _
                                            True, oCompDef.WorkPoints.Item(1))

    ' Change the name.
    oSketch.Name = "My Origin Sketch"

    ' Draw a circle at the origin of the sketch plane.
    oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 1)

    ' Draw a line along the X axis.
    Call oSketch.SketchLines.AddByTwoPoints(oCircle.CenterSketchPoint, _
                ThisApplication.TransientGeometry.CreatePoint2d(1, 0))
```

Querying a sketch profile to get regions.

Description

This sample demonstrates getting region properties from a sketch profile.

Code Samples

- VBA
- iLogic

To run the sample you must have a sketch active that contains some sketch entities.

Copy Code

```
Public Sub GetProfileRegionProperties()
    ' Set a reference to the active sketch.
    ' This assumes a 2D sketch is active.
    Dim oSketch As Sketch
    Set oSketch = ThisApplication.ActiveEditObject

    ' Create a default profile from the sketch.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Set a reference to the region properties object.
    Dim oRegionProps As RegionProperties
    Set oRegionProps = oProfile.RegionProperties

    ' Set the accuracy to medium.
    oRegionProps.Accuracy = kMedium

    ' Display the region properties of the profile.
    Debug.Print "Area: " & oRegionProps.Area

    Debug.Print "Perimeter: " & oRegionProps.Perimeter

    Debug.Print "Centroid: " & _
                oRegionProps.Centroid.X & ", " & _
                oRegionProps.Centroid.Y

    Dim adPrincipalMoments(1 To 3) As Double
    Call oRegionProps.PrincipalMomentsOfInertia(adPrincipalMoments(1), _
                                                adPrincipalMoments(2), _
                                                adPrincipalMoments(3))
    Debug.Print "Principal Moments of Inertia: " & _
                adPrincipalMoments(1) & ", " & _
                adPrincipalMoments(2)

    Dim adRadiusOfGyration(1 To 3) As Double
    Call oRegionProps.RadiusOfGyration(adRadiusOfGyration(1), _
                                       adRadiusOfGyration(2), _
                                       adRadiusOfGyration(3))
    Debug.Print "Radius of Gyration: " & _
                adRadiusOfGyration(1) & ", " & _
                adRadiusOfGyration(2)

    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oRegionProps.MomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Debug.Print "Moments: "
    Debug.Print " Ixx: " & Ixx
    Debug.Print " Iyy: " & Iyy
    Debug.Print " Ixy: " & Ixy

    Debug.Print "Rotation Angle from projected Sketch Origin to Principle Axes: " _
    & oRegionProps.RotationAngle

End Sub
```

To run the sample you must have a sketch active that contains some sketch entities.

Copy Code

```
    ' Set a reference to the active sketch.
    ' This assumes a 2D sketch is active.
    Dim oSketch As Sketch
    oSketch = ThisApplication.ActiveEditObject

    ' Create a default profile from the sketch.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Set a reference to the region properties object.
    Dim oRegionProps As RegionProperties
    oRegionProps = oProfile.RegionProperties

    ' Set the accuracy to medium.
    oRegionProps.Accuracy = kMedium

    ' Display the region properties of the profile.
    Logger.Info("Area: " & oRegionProps.Area)


    Logger.Info("Perimeter: " & oRegionProps.Perimeter)


    Logger.Info("Centroid: " & _
```

```
                oRegionProps.Centroid.X & ", " & _
                oRegionProps.Centroid.Y)

    Dim adPrincipalMoments(0 To 2) As Double
    Call oRegionProps.PrincipalMomentsOfInertia(adPrincipalMoments(0), _
                                                adPrincipalMoments(1), _
                                                adPrincipalMoments(2))
    Logger.Info("Principal Moments of Inertia: " & _
                adPrincipalMoments(1) & ", " & _
                adPrincipalMoments(2))

    Dim adRadiusOfGyration(0 To 2) As Double
    Call oRegionProps.RadiusOfGyration(adRadiusOfGyration(0), _
                                       adRadiusOfGyration(1), _
                                       adRadiusOfGyration(2))
    Logger.Info("Radius of Gyration: " & _
                adRadiusOfGyration(0) & ", " & _
                adRadiusOfGyration(1))

    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oRegionProps.MomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Logger.Info("Moments: ")

    Logger.Info(" Ixx: " & Ixx)

    Logger.Info(" Iyy: " & Iyy)

    Logger.Info(" Ixy: " & Ixy)


    Logger.Info("Rotation Angle from projected Sketch Origin to Principle Axes: " _
    & oRegionProps.RotationAngle)
```

Sketch profile control

Description

This sample demonstrates the usage of the Profiles API to control the shape of the profile. The sample creates three concntric circles and creates an extrusion of the region between the inner circles.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SketchProfileControl()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane. Since it's being created on
    ' one of the base workplanes we know the orientation it will be created in
    ' and don't need to worry about controlling it. Because of this we also
    ' know the origin of the sketch plane will be at (0,0,0) in model space.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw 3 concentric circles.

    Dim oCircle1 As SketchCircle
    Set oCircle1 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 6)

    Dim oCircle2 As SketchCircle
    Set oCircle2 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 4)

    Dim oCircle3 As SketchCircle
    Set oCircle3 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 2)

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Modify the profile: the returned profile consists of 3
    ' paths each corresponding to a sketch circle. The desired
    ' result is that the innermost path removes material and the
    ' second path adds material. The outermost path is not needed
    ' and is hence deleted.
    Dim oProfPath As ProfilePath
    For Each oProfPath In oProfile
```

```
      If oProfPath.Item(1).SketchEntity Is oCircle3 Then
        oProfPath.AddsMaterial = False
      ElseIf oProfPath.Item(1).SketchEntity Is oCircle2 Then
        oProfPath.AddsMaterial = False
      Else
        oProfPath.Delete
      End If
    Next

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    'Make the sketch visible for better visualization
    oSketch.Visible = True
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane. Since it's being created on
    ' one of the base workplanes we know the orientation it will be created in
    ' and don't need to worry about controlling it. Because of this we also
    ' know the origin of the sketch plane will be at (0,0,0) in model space.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw 3 concentric circles.

    Dim oCircle1 As SketchCircle
    oCircle1 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 6)

    Dim oCircle2 As SketchCircle
    oCircle2 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 4)

    Dim oCircle3 As SketchCircle
    oCircle3 = oSketch.SketchCircles.AddByCenterRadius( _
    oTransGeom.CreatePoint2d(5, 5), 2)

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Modify the profile: the returned profile consists of 3
    ' paths each corresponding to a sketch circle. The desired
    ' result is that the innermost path removes material and the
    ' second path adds material. The outermost path is not needed
    ' and is hence deleted.
    Dim oProfPath As ProfilePath
    For Each oProfPath In oProfile
      If oProfPath.Item(1).SketchEntity Is oCircle3 Then
        oProfPath.AddsMaterial = False
      ElseIf oProfPath.Item(1).SketchEntity Is oCircle2 Then
        oProfPath.AddsMaterial = False
      Else
        oProfPath.Delete
      End If
    Next

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    'Make the sketch visible for better visualization
    oSketch.Visible = True
```

Projection - project across parts

Description

This sample demonstrates projecting a sketch entity across parts in an assembly. To use the sample, have an assembly open that contains at least two occurrences, (parts only), and run the program.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ProjectingAcrossParts()
    ' Set a reference to the assembly component definition.
    Dim oAsmCompDef As AssemblyComponentDefinition
    Set oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get references to two occurrences.
    ' This arbitrarily gets the first and second occurrence.
    Dim oOcc1 As ComponentOccurrence
    Set oOcc1 = oAsmCompDef.Occurrences.Item(1)

    Dim oOcc2 As ComponentOccurrence
    Set oOcc2 = oAsmCompDef.Occurrences.Item(2)

    ' Create a sketch on the first part.
    Dim oSketch1 As PlanarSketch
    Set oSketch1 = oOcc1.Definition.Sketches.Add(oOcc1.Definition.WorkPlanes(1))

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a sketch line on the sketch.
    Dim oSketchLine1 As Object
    Set oSketchLine1 = oSketch1.SketchLines.AddByTwoPoints(oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 0))

    ' Because we need the sketch line in the context of the assembly
    ' we need to create a proxy for the sketch line. The proxy
    ' represents the sketch line in the context of the assembly.
    Dim oSketchLine1Proxy As Object
    Call oOcc1.CreateGeometryProxy(oSketchLine1, oSketchLine1Proxy)

    ' Create a sketch on the second part.
    Dim oSketch2 As PlanarSketch
    Set oSketch2 = oOcc2.Definition.Sketches.Add(oOcc2.Definition.WorkPlanes(1))

    ' Create a proxy for the sketch in the second part.
    Dim oSketch2Proxy As PlanarSketchProxy
    Call oOcc2.CreateGeometryProxy(oSketch2, oSketch2Proxy)

    ' Project the line in the sketch in the first
    ' part to the sketch in the second part
    Dim oSketchLine2Proxy As SketchLineProxy
    Set oSketchLine2Proxy = oSketch2Proxy.AddByProjectingEntity(oSketchLine1Proxy)
End Sub
```

Copy Code

```
    ' Set a reference to the assembly component definition.
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Get references to two occurrences.
    ' This arbitrarily gets the first and second occurrence.
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDef.Occurrences.Item(1)

    Dim oOcc2 As ComponentOccurrence
    oOcc2 = oAsmCompDef.Occurrences.Item(2)

    ' Create a sketch on the first part.
    Dim oSketch1 As PlanarSketch
    oSketch1 = oOcc1.Definition.Sketches.Add(oOcc1.Definition.WorkPlanes(1))

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a sketch line on the sketch.
    Dim oSketchLine1 As Object
    oSketchLine1 = oSketch1.SketchLines.AddByTwoPoints(oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 0))

    ' Because we need the sketch line in the context of the assembly
    ' we need to create a proxy for the sketch line. The proxy
    ' represents the sketch line in the context of the assembly.
    Dim oSketchLine1Proxy As Object
    Call oOcc1.CreateGeometryProxy(oSketchLine1, oSketchLine1Proxy)

    ' Create a sketch on the second part.
    Dim oSketch2 As PlanarSketch
    oSketch2 = oOcc2.Definition.Sketches.Add(oOcc2.Definition.WorkPlanes(1))

    ' Create a proxy for the sketch in the second part.
    Dim oSketch2Proxy As PlanarSketchProxy
    Call oOcc2.CreateGeometryProxy(oSketch2, oSketch2Proxy)

    ' Project the line in the sketch in the first
    ' part to the sketch in the second part
    Dim oSketchLine2Proxy As SketchLineProxy
    oSketchLine2Proxy = oSketch2Proxy.AddByProjectingEntity(oSketchLine1Proxy)
```

Copy sketch contents

Description

This sample shows how to copy the contents of one sketch to another.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CopySketch()
    Dim oCmdMgr As CommandManager
    Set oCmdMgr = ThisApplication.CommandManager

    Dim oSourceSketch As Sketch
    Set oSourceSketch = oCmdMgr.Pick(kSketchObjectFilter, "Select source sketch")

    Dim oTargetSketch As Sketch
    Set oTargetSketch = oCmdMgr.Pick(kSketchObjectFilter, "Select target sketch")

    Call oSourceSketch.CopyContentsTo(oTargetSketch)
End Sub
```

Copy Code

```
    Dim oCmdMgr As CommandManager
    oCmdMgr = ThisApplication.CommandManager

    Dim oSourceSketch As Sketch
    oSourceSketch = oCmdMgr.Pick(kSketchObjectFilter, "Select source sketch")

    Dim oTargetSketch As Sketch
    oTargetSketch = oCmdMgr.Pick(kSketchObjectFilter, "Select target sketch")

    Call oSourceSketch.CopyContentsTo(oTargetSketch)
```

Defer sketch updates

Description

This sample demonstrates the sketch defer update functionality.

Code Samples

- VBA
- iLogic

To run this sample, have a 2d sketch active. Choosing 'Yes' in the dialog popped up by this sample defers the sketch update and creates the circle (approximated by 500 lines) much faster.

Copy Code

```
Public Sub DeferSketchUpdate()
   ' Check to make sure a sketch is open.
   If Not Typeof ThisApplication.ActiveEditObject Is Sketch Then
      MsgBox "A sketch must be active."
      Exit Sub
   End If

   ' Set a reference to the active sketch.
   Dim oSketch As Sketch
   Set oSketch = ThisApplication.ActiveEditObject

   ' Set a reference to the transient geometry collection.
   Dim oTransGeom As TransientGeometry
   Set oTransGeom = ThisApplication.TransientGeometry

   Dim bDeferUpdatesSet As Boolean
   bDeferUpdatesSet = False

   If MsgBox("Suppress sketch solve?", vbYesNo) = vbYes Then
      ' Defer sketch updates
      oSketch.DeferUpdates = True
      bDeferUpdatesSet = True
   End If

   ' Draw a circle using 500 lines.
   Dim dRadius As Double
   dRadius = 2
   Dim dAngle As Double
   dAngle = 0
   Dim i As Long
   For i = 0 To 499
      Dim oPt1 As Point2d
      Set oPt1 = oTransGeom.CreatePoint2d(dRadius * Cos(dAngle), dRadius * Sin(dAngle))

      dAngle = dAngle + (3.14159265358979 / 250)

      Dim oPt2 As Point2d
      Set oPt2 = oTransGeom.CreatePoint2d(dRadius * Cos(dAngle), dRadius * Sin(dAngle))

      Dim oLine As SketchLine
      Set oLine = oSketch.SketchLines.AddByTwoPoints(oPt1, oPt2)

   Next

   If bDeferUpdatesSet = True Then
      ' Unset defer updates
```

```
      oSketch.DeferUpdates = False 'A full sketch solve occurs
   End If
End Sub
```

To run this sample, have a 2d sketch active. Choosing 'Yes' in the dialog popped up by this sample defers the sketch update and creates the circle (approximated by 500 lines) much faster.

Copy Code

```
    ' Check to make sure a sketch is open.
   If Not Typeof ThisApplication.ActiveEditObject Is Sketch Then
      MsgBox("A sketch must be active.")
      Exit Sub
   End If

   ' Set a reference to the active sketch.
   Dim oSketch As Sketch
   oSketch = ThisApplication.ActiveEditObject

   ' Set a reference to the transient geometry collection.
   Dim oTransGeom As TransientGeometry
   oTransGeom = ThisApplication.TransientGeometry

   Dim bDeferUpdatesSet As Boolean
   bDeferUpdatesSet = False

   If MsgBox("Suppress sketch solve?", vbYesNo) = vbYes Then
      ' Defer sketch updates
      oSketch.DeferUpdates = True
      bDeferUpdatesSet = True
   End If

   ' Draw a circle using 500 lines.
   Dim dRadius As Double
   dRadius = 2
   Dim dAngle As Double
   dAngle = 0
   Dim i As Long
   For i = 0 To 499
      Dim oPt1 As Point2d
      oPt1 = oTransGeom.CreatePoint2d(dRadius * Cos(dAngle), dRadius * Sin(dAngle))

      dAngle = dAngle + (3.14159265358979 / 250)

      Dim oPt2 As Point2d
      oPt2 = oTransGeom.CreatePoint2d(dRadius * Cos(dAngle), dRadius * Sin(dAngle))

      Dim oLine As SketchLine
      oLine = oSketch.SketchLines.AddByTwoPoints(oPt1, oPt2)

   Next

   If bDeferUpdatesSet = True Then
      ' Unset defer updates
      oSketch.DeferUpdates = False 'A full sketch solve occurs
   End If
```

Sketch Delete

Description

This sample demonstrates deleting a sketch.

Code Samples

- VBA
- iLogic

To use this sample, have a part document open that contains a sketch named Sketch1. The sketch cannot have any dependents, or the delete will fail.

Copy Code

```
Public Sub DeleteSketch()
    ' Set a reference to the Sketches collection. This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    Set oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get the sketch named "Sketch1".
    On Error Resume Next
    Dim oSketch As PlanarSketch
    Set oSketch = oSketches.Item("Sketch1")
    If Err Then
      MsgBox "A Sketch named ""Sketch1"" must exist."
      Exit Sub
    End If
    On Error GoTo 0

    ' Check to see if the sketch has any dependents.
    If oSketch.Dependents.Count > 0 Then
      MsgBox "Cannot delete a sketch with dependents."
    Else
      ' Delete the sketch.
      oSketch.Delete
    End If
End Sub
```

To use this sample, have a part document open that contains a sketch named Sketch1. The sketch cannot have any dependents, or the delete will fail.

Copy Code

```
    ' Set a reference to the Sketches collection. This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get the sketch named "Sketch1".
    On Error Resume Next
    Dim oSketch As PlanarSketch
    oSketch = oSketches.Item("Sketch1")
    If Err.Number > 0 Then
      MsgBox("A Sketch named ""Sketch1"" must exist.")
      Exit Sub
    End If
    On Error GoTo 0

    ' Check to see if the sketch has any dependents.
    If oSketch.Dependents.Count > 0 Then
      MsgBox("Cannot delete a sketch with dependents.")
    Else
      ' Delete the sketch.
      oSketch.Delete
    End If
```

Sketch Open for Edit

Description

This sample demonstrates opening a sketch for edit.

Code Samples

- VBA
- iLogic

To use this sample, have a part document open that contains a sketch named Sketch1.

Copy Code

```
Public Sub StartSketchEdit()
    ' Set a reference to the Sketches collection. This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    Set oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get the sketch named "Sketch1".
    On Error Resume Next
    Dim oSketch As PlanarSketch
    Set oSketch = oSketches.Item("Sketch1")
    If Err Then
      MsgBox "A Sketch named ""Sketch1"" must exist."
      Exit Sub
    End If
    On Error GoTo 0

    ' Open the sketch for edit to allow the user to edit it.
    oSketch.Edit
End Sub
```

To use this sample, have a part document open that contains a sketch named Sketch1.

Copy Code

```
    ' Set a reference to the Sketches collection. This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get the sketch named "Sketch1".
    On Error Resume Next
    Dim oSketch As PlanarSketch
    oSketch = oSketches.Item("Sketch1")
    If Err.Number > 0 Then
      MsgBox("A Sketch named ""Sketch1"" must exist.")
      Exit Sub
    End If
    On Error GoTo 0

    ' Open the sketch for edit to allow the user to edit it.
    oSketch.Edit
```

Move sketch entities

Description

This sample demonstrates the translation of all the objects on the active sketch by a certain distance.

Code Samples

- VBA
- iLogic

Have a sketch active that contains several entities (this could include text boxes and images) and run the sample.

Copy Code

```
Public Sub MoveSketchObjects()
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is Sketch Then
        MsgBox "A sketch must be active."
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As Sketch
    Set oSketch = ThisApplication.ActiveEditObject

    ' Create a vector along the x-axis.
    Dim oVec As Vector2d
    Set oVec = ThisApplication.TransientGeometry.CreateVector2d(5, 0)

    Dim oSketchObjects As ObjectCollection
    Set oSketchObjects = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all entities in the sketch
    Dim oSketchEntity As SketchEntity
    For Each oSketchEntity In oSketch.SketchEntities
        oSketchObjects.Add oSketchEntity
    Next

    ' Move all sketch objects along x-axis by 5 units.
    ' This will move all the text boxes and images in
    ' the sketch as well since these have sketch lines
    ' as boundary geometry or a sketch point as an
    ' origin point.
    Call oSketch.MoveSketchObjects(oSketchObjects, oVec)
End Sub
```

Have a sketch active that contains several entities (this could include text boxes and images) and run the sample.

Copy Code

```
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is Sketch Then
        MsgBox("A sketch must be active.")
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As Sketch
    oSketch = ThisApplication.ActiveEditObject

    ' Create a vector along the x-axis.
    Dim oVec As Vector2d
    oVec = ThisApplication.TransientGeometry.CreateVector2d(5, 0)

    Dim oSketchObjects As ObjectCollection
    oSketchObjects = ThisApplication.TransientObjects.CreateObjectCollection

    ' Get all entities in the sketch
    Dim oSketchEntity As SketchEntity
    For Each oSketchEntity In oSketch.SketchEntities
        oSketchObjects.Add(oSketchEntity)
    Next

    ' Move all sketch objects along x-axis by 5 units.
    ' This will move all the text boxes and images in
    ' the sketch as well since these have sketch lines
    ' as boundary geometry or a sketch point as an
    ' origin point.
    Call oSketch.MoveSketchObjects(oSketchObjects, oVec)
```

Offset a 2D sketch

Description

This sample demonstrates the creation of offsets in 2d sketches. Two ways of creating the offset are shown - one uses a distance and the other uses the input point.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub Offset()
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox "A sketch must be active."
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = ThisApplication.ActiveEditObject
```

```
    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a rectangle
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                                            oTransGeom.CreatePoint2d(0, 0), _
                                            oTransGeom.CreatePoint2d(10, -10))

    ' Create a new object collection
    Dim oCollection As ObjectCollection
    Set oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add the first sketch line of the rectangle to the collection
    oCollection.Add oRectangleLines.Item(1)

    ' Create a point at (0,3). The entity resulting
    ' from the offset of the first sketch line must
    ' pass thru this point.
    Dim oOffsetPoint As Point2d
    Set oOffsetPoint = oTransGeom.CreatePoint2d(0, 3)

    ' Create an offset rectangle using the offset point.
    Call oSketch.OffsetSketchEntitiesUsingPoint(oCollection, oOffsetPoint, True)

    ' Get the sketch normal
    Dim oNormalVector As UnitVector
    Set oNormalVector = oSketch.PlanarEntityGeometry.Normal

    ' Get the direction of the sketch line being offset
    Dim oLineDir As UnitVector2d
    Set oLineDir = oRectangleLines.Item(1).Geometry.Direction

    Dim oLineVector As UnitVector
    Set oLineVector = oTransGeom.CreateUnitVector(oLineDir.X, oLineDir.Y, 0)

    ' The cross product of these vectors is the
    ' natural offset direction for the sketch line.
    Dim oOffsetVector As UnitVector
    Set oOffsetVector = oLineVector.CrossProduct(oNormalVector)

    ' Get the desired offset vector (the +ve y-axis)
    Dim oDesiredVector As UnitVector
    Set oDesiredVector = oTransGeom.CreateUnitVector(0, 1, 0)

    Dim bNaturalOffsetDir As Boolean

    If oOffsetVector.IsEqualTo(oDesiredVector) Then
        bNaturalOffsetDir = True
    Else
        bNaturalOffsetDir = False
    End If

    ' Create an offset at a distance of 6 cms.
    Call oSketch.OffsetSketchEntitiesUsingDistance(oCollection, 6, bNaturalOffsetDir, True)
End Sub
```

Copy Code

```
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox("A sketch must be active.")
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a rectangle
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                                            oTransGeom.CreatePoint2d(0, 0), _
                                            oTransGeom.CreatePoint2d(10, -10))

    ' Create a new object collection
    Dim oCollection As ObjectCollection
    oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add the first sketch line of the rectangle to the collection
    oCollection.Add(oRectangleLines.Item(1))

    ' Create a point at (0,3). The entity resulting
    ' from the offset of the first sketch line must
    ' pass thru this point.
    Dim oOffsetPoint As Point2d
    oOffsetPoint = oTransGeom.CreatePoint2d(0, 3)

    ' Create an offset rectangle using the offset point.
    Call oSketch.OffsetSketchEntitiesUsingPoint(oCollection, oOffsetPoint, True)

    ' Get the sketch normal
    Dim oNormalVector As UnitVector
    oNormalVector = oSketch.PlanarEntityGeometry.Normal

    ' Get the direction of the sketch line being offset
    Dim oLineDir As UnitVector2d
    oLineDir = oRectangleLines.Item(1).Geometry.Direction

    Dim oLineVector As UnitVector
    oLineVector = oTransGeom.CreateUnitVector(oLineDir.X, oLineDir.Y, 0)
```

```
    ' The cross product of these vectors is the
    ' natural offset direction for the sketch line.
    Dim oOffsetVector As UnitVector
    oOffsetVector = oLineVector.CrossProduct(oNormalVector)

    ' Get the desired offset vector (the +ve y-axis)
    Dim oDesiredVector As UnitVector
    oDesiredVector = oTransGeom.CreateUnitVector(0, 1, 0)

    Dim bNaturalOffsetDir As Boolean

    If oOffsetVector.IsEqualTo(oDesiredVector) Then
        bNaturalOffsetDir = True
    Else
        bNaturalOffsetDir = False
    End If

    ' Create an offset at a distance of 6 cms.
    Call oSketch.OffsetSketchEntitiesUsingDistance(oCollection, 6, bNaturalOffsetDir, True)
```

Sketch Lines

Description

This sample demonstrates creating lines. It uses all of the various methods to create lines, both singly and as rectangles.

Code Samples

- VBA
- iLogic

To run this sample you must have a sketch active.

Copy Code

```
Public Sub DrawSketchLine()
    ' Check to make sure a sketch is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is PlanarSketch Then
      MsgBox "A sketch must be active."
      Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a new transaction to wrap the construction of the three lines
    ' into a single undo.
    Dim oTrans As Transaction
    Set oTrans = ThisApplication.TransactionManager.StartTransaction( _
    ThisApplication.ActiveDocument, _
    "Create Triangle Sample")

    ' Create the first line of the triangle. This uses two transient points as
    ' input to definethe coordinates of the ends of the line. Since a transient
    ' point is input a sketch point is automatically created at that location and
    ' the line is attached to it.
    Dim oLines(1 To 3) As SketchLine
    Set oLines(1) = oSketch.SketchLines.AddByTwoPoints(oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 0))

    ' Create a sketch line that is connected to the sketch point the previous lines
    ' end point is connected to. This will automatically create the constraint to
    ' tie the new line to the sketch point the previous line is also connected to.
    ' This will result in the the two lines being connected since they're both tied
    ' to the same sketch point.
    Set oLines(2) = oSketch.SketchLines.AddByTwoPoints(oLines(1).EndSketchPoint, _
    oTransGeom.CreatePoint2d(2, 3))

    ' Create a third line and connect it to the start point of the first line and the
    ' end point of the second line. This will result in a connected triangle.
    Set oLines(3) = oSketch.SketchLines.AddByTwoPoints(oLines(2).EndSketchPoint, _
    oLines(1).StartSketchPoint)

    ' End the transaction for the triangle.
    oTrans.End

    ' Create a rectangle whose lines are parallel to the sketch planes x and y axes
    ' by using the SketchLines.AddAsTwoPointRectangle method. The top point of the
    ' triangle will be used as input for one of the points so the rectangle will be
    ' tied to that point.
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oLines(3).StartSketchPoint, _
    oTransGeom.CreatePoint2d(5, 5))

    ' Create a rotated rectangle by using the SketchLines.AddAsTwoRectangle method.
    ' One of the corners of this rectangle will be tied to the corner of the previous
    ' rectangle.
    Set oRectangleLines = oSketch.SketchLines.AddAsThreePointRectangle( _
    oRectangleLines(2).EndSketchPoint, _
    oTransGeom.CreatePoint2d(7, 3), _
    oTransGeom.CreatePoint2d(8, 8))
End Sub
```

To run this sample you must have a sketch active.

Copy Code

```
    ' Check to make sure a sketch is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is PlanarSketch Then
      MsgBox("A sketch must be active.")
      Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a new transaction to wrap the construction of the three lines
    ' into a single undo.
    Dim oTrans As Transaction
    oTrans = ThisApplication.TransactionManager.StartTransaction( _
    ThisApplication.ActiveDocument, _
    "Create Triangle Sample")

    ' Create the first line of the triangle. This uses two transient points as
    ' input to definethe coordinates of the ends of the line. Since a transient
    ' point is input a sketch point is automatically created at that location and
    ' the line is attached to it.
    Dim oLines(0 To 2) As SketchLine
    oLines(0) = oSketch.SketchLines.AddByTwoPoints(oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 0))

    ' Create a sketch line that is connected to the sketch point the previous lines
    ' end point is connected to. This will automatically create the constraint to
    ' tie the new line to the sketch point the previous line is also connected to.
    ' This will result in the the two lines being connected since they're both tied
    ' to the same sketch point.
    oLines(1) = oSketch.SketchLines.AddByTwoPoints(oLines(0).EndSketchPoint, _
    oTransGeom.CreatePoint2d(2, 3))

    ' Create a third line and connect it to the start point of the first line and the
    ' end point of the second line. This will result in a connected triangle.
    oLines(2) = oSketch.SketchLines.AddByTwoPoints(oLines(1).EndSketchPoint, _
    oLines(0).StartSketchPoint)

    ' End the transaction for the triangle.
    oTrans.End

    ' Create a rectangle whose lines are parallel to the sketch planes x and y axes
    ' by using the SketchLines.AddAsTwoPointRectangle method. The top point of the
    ' triangle will be used as input for one of the points so the rectangle will be
    ' tied to that point.
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oLines(2).StartSketchPoint, _
    oTransGeom.CreatePoint2d(5, 5))

    ' Create a rotated rectangle by using the SketchLines.AddAsTwoRectangle method.
    ' One of the corners of this rectangle will be tied to the corner of the previous
    ' rectangle.
    oRectangleLines = oSketch.SketchLines.AddAsThreePointRectangle( _
    oRectangleLines(2).EndSketchPoint, _
    oTransGeom.CreatePoint2d(7, 3), _
    oTransGeom.CreatePoint2d(8, 8))
```

Set Sketch Visibility

Description

This sample demonstrates setting the visibility of a sketch.

Code Samples

- VBA
- iLogic

To use this sample have a part document open that contains at least one sketch.

Copy Code

```
Public Sub ToggleSketchVisibility()
    ' Set a reference to the Sketches collection.  This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    Set oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get whether the sketch visibility should be turned on or off.
    Dim bVisibleOn As Boolean
    If MsgBox("Do you want to turn all sketches on?", vbYesNo + vbQuestion) = vbYes Then
        bVisibleOn = True
    Else
        bVisibleOn = False
    End If

    ' Iterate through all of the sketches and set their visibility.
    Dim oSketch As PlanarSketch
    For Each oSketch In oSketches
```

```
        If bVisibleOn Then
            oSketch.Visible = True
        Else
            oSketch.Visible = False
        End If
    Next
End Sub
```

To use this sample have a part document open that contains at least one sketch.

Copy Code

```
    ' Set a reference to the Sketches collection.  This assumes
    ' that a part document containing a sketch is active.
    Dim oSketches As PlanarSketches
    oSketches = ThisApplication.ActiveDocument.ComponentDefinition.Sketches

    ' Get whether the sketch visibility should be turned on or off.
    Dim bVisibleOn As Boolean
    If MsgBox("Do you want to turn all sketches on?", vbYesNo + vbQuestion) = vbYes Then
        bVisibleOn = True
    Else
        bVisibleOn = False
    End If

    ' Iterate through all of the sketches and set their visibility.
    Dim oSketch As PlanarSketch
    For Each oSketch In oSketches
        If bVisibleOn Then
            oSketch.Visible = True
        Else
            oSketch.Visible = False
        End If
    Next
```

Create and insert a sketch block definition into a part sketch

Description

This sample demonstrates inserting a sketch block into a part sketch.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSketchBlockDefinition()
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Create the new sketch block definition.
    Dim oSketchBlockDef As SketchBlockDefinition
    Set oSketchBlockDef = oPartDoc.ComponentDefinition.SketchBlockDefinitions.Add("My Block Def")

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketchBlockDef.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

End Sub

Public Sub InsertSketchBlockDefinition()

    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the definition named "My Block Def"
    Dim oSketchBlockDef As SketchBlockDefinition
    Set oSketchBlockDef = oCompDef.SketchBlockDefinitions.Item("My Block Def")

    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(10, 10)

    ' Insert the sketch block definition
    Call oSketch.SketchBlocks.AddByDefinition(oSketchBlockDef, oPosition)
End Sub
```

Copy Code

```
Sub Main
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Create the new sketch block definition.
    Dim oSketchBlockDef As SketchBlockDefinition
    oSketchBlockDef = oPartDoc.ComponentDefinition.SketchBlockDefinitions.Add("My Block Def")

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketchBlockDef.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

End Sub

Public Sub InsertSketchBlockDefinition()

    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the definition named "My Block Def"
    Dim oSketchBlockDef As SketchBlockDefinition
    oSketchBlockDef = oCompDef.SketchBlockDefinitions.Item("My Block Def")

    Dim oPosition As Point2d
    oPosition = ThisApplication.TransientGeometry.CreatePoint2d(10, 10)

    ' Insert the sketch block definition
    Call oSketch.SketchBlocks.AddByDefinition(oSketchBlockDef, oPosition)
End Sub
```

Create sketch block from an existing sketch

Description

This sample demonstrates creating a sketch block from an existing sketch.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSketchBlock()
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    Dim oSketchObjects As ObjectCollection
    Set oSketchObjects = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oLine As SketchEntity
    For Each oLine In oRectangleLines
       Call oSketchObjects.Add(oLine)
    Next

    ' Insert the sketch block definition
    Call oSketch.SketchBlocks.Add(oSketchObjects)
End Sub
```

Copy Code

```
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    Dim oSketchObjects As ObjectCollection
    oSketchObjects = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oLine As SketchEntity
    For Each oLine In oRectangleLines
       Call oSketchObjects.Add(oLine)
    Next

    ' Insert the sketch block definition
    Call oSketch.SketchBlocks.Add(oSketchObjects)
```

Create sketch elliptical arc

Description

This sample demonstrates creating an elliptical arc in a sketch and dimensioning its minor radius.

Code Samples

- VBA
- iLogic

To run this sample you must have a sketch active.

Copy Code

```
Sub DrawSketchEllipticalArc()
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox "A sketch must be active."
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Arc center point at 0,0
    Dim oCenter As Point2d
    Set oCenter = oTransGeom.CreatePoint2d(0, 0)

    ' Arc major axis as the y-axis
    Dim oMajorAxis As UnitVector2d
    Set oMajorAxis = oTransGeom.CreateUnitVector2d(0, 1)

    Dim PI As Double
    PI = 4# * Atn(1#)

    ' Create the elliptical arc
    Dim oEllipticalArc As SketchEllipticalArc
    Set oEllipticalArc = oSketch.SketchEllipticalArcs.Add(oCenter, oMajorAxis, 5, 2, 0, PI / 2)

    Dim oTextPt As Point2d
    Set oTextPt = oTransGeom.CreatePoint2d(-1, -0.5)

    ' Create a radius dimension
    Dim oEllipseRadiusDim As EllipseRadiusDimConstraint
    Set oEllipseRadiusDim = oSketch.DimensionConstraints.AddEllipseRadius(oEllipticalArc, False, oTextPt)
End Sub
```

To run this sample you must have a sketch active.

Copy Code

```
    ' Check to make sure a sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox("A sketch must be active.")
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    oSketch = ThisApplication.ActiveEditObject
```

```
    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Arc center point at 0,0
    Dim oCenter As Point2d
    oCenter = oTransGeom.CreatePoint2d(0, 0)

    ' Arc major axis as the y-axis
    Dim oMajorAxis As UnitVector2d
    oMajorAxis = oTransGeom.CreateUnitVector2d(0, 1)

    Dim PI As Double
    PI = 4# * Math.Atan(1#)

    ' Create the elliptical arc
    Dim oEllipticalArc As SketchEllipticalArc
    oEllipticalArc = oSketch.SketchEllipticalArcs.Add(oCenter, oMajorAxis, 5, 2, 0, PI / 2)

    Dim oTextPt As Point2d
    oTextPt = oTransGeom.CreatePoint2d(-1, -0.5)

    ' Create a radius dimension
    Dim oEllipseRadiusDim As EllipseRadiusDimConstraint
    oEllipseRadiusDim = oSketch.DimensionConstraints.AddEllipseRadius(oEllipticalArc, False, oTextPt)
```

Sketch Display Entities

Description

This sample demonstrates the query functionality available for sketch entities.

Code Samples

- VBA
- iLogic

To use this sample, select a sketch entity that you want to query. Run the program and it will display information about the selected entity in the immediate window.

Copy Code

```
Public Sub SketchEntities()
    ' Get the first item in the select set.  This assumes
    ' something is selected and that it's a sketch entity.
    Dim oSketchEnt As SketchEntity
    Set oSketchEnt = ThisApplication.ActiveDocument.SelectSet.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Display type specific information.
    Select Case oSketchEnt.Type
        Case kSketchArcObject
            Dim oArc As SketchArc
            Set oArc = oSketchEnt
            Debug.Print "Sketch Arc selected."
            Debug.Print "  Center Point: " & oArc.CenterSketchPoint.Geometry.X & _
                        ", " & oArc.CenterSketchPoint.Geometry.Y
            Debug.Print "  Start Point: " & oArc.StartSketchPoint.Geometry.X & _
                        ", " & oArc.StartSketchPoint.Geometry.Y
            Debug.Print "  End Point: " & oArc.EndSketchPoint.Geometry.X & _
                        ", " & oArc.EndSketchPoint.Geometry.Y
            Debug.Print "  Start Angle: " & oArc.StartAngle
            Debug.Print "  Sweep Angle: " & oArc.SweepAngle
            Debug.Print "  Radius: " & oArc.Radius
            Debug.Print "  Length: " & oArc.Length
        Case kSketchCircleObject
            Dim oCircle As SketchCircle
            Set oCircle = oSketchEnt
            Debug.Print "Sketch Circle selected."
            Debug.Print "  Center Point: " & oCircle.CenterSketchPoint.Geometry.X & _
                        ", " & oCircle.CenterSketchPoint.Geometry.Y
            Debug.Print "  Radius: " & oCircle.Radius
            Debug.Print "  Area: " & oCircle.Area

            ' Change the radius
            oCircle.Radius = oCircle.Radius * 1.5
        Case kSketchEllipseObject
            Dim oEllipse As SketchEllipse
            Set oEllipse = oSketchEnt
            Debug.Print "Sketch Ellipse selected."
            Debug.Print "  Center Point: " & oEllipse.CenterSketchPoint.Geometry.X & _
                        ", " & oEllipse.CenterSketchPoint.Geometry.Y
            Debug.Print "  Major Axis Vecotr: " & oEllipse.MajorAxisVector.X & _
                        ", " & oEllipse.MajorAxisVector.Y
            Debug.Print "  Major Radius: " & oEllipse.MajorRadius
            Debug.Print "  Minor Radius: " & oEllipse.MinorRadius
            Debug.Print "  Area: " & oEllipse.Area

            ' Modify the ellipse
            oEllipse.MajorAxisVector = oTransGeom.CreateUnitVector2d(1, 1)
            oEllipse.MajorRadius = oEllipse.MajorRadius / 2
            oEllipse.MinorRadius = oEllipse.MinorRadius * 2
        Case kSketchEllipticalArcObject
            Dim oEllipticalArc As SketchEllipticalArc
            Set oEllipticalArc = oSketchEnt
            Debug.Print "Sketch Elliptical Arc selected."
```

```
            Debug.Print "  Center Point: " & oEllipticalArc.CenterSketchPoint.Geometry.X & _
                        ", " & oEllipticalArc.CenterSketchPoint.Geometry.Y
            Debug.Print "  Major Axis Vector: " & oEllipticalArc.MajorAxisVector.X & _
                        ", " & oEllipticalArc.MajorAxisVector.Y
            Debug.Print "  Major Radius: " & oEllipticalArc.MajorRadius
            Debug.Print "  Minor Radius: " & oEllipticalArc.MinorRadius
            Debug.Print "  Start Angle: " & oEllipticalArc.StartAngle
            Debug.Print "  Sweep Angle: " & oEllipticalArc.SweepAngle
            Debug.Print "  Length: " & oEllipticalArc.Length

            ' Modify the elliptical arc.
            oEllipticalArc.MajorAxisVector = oTransGeom.CreateUnitVector2d(1, 1)
            oEllipticalArc.MajorRadius = oEllipticalArc.MajorRadius / 2
            oEllipticalArc.MinorRadius = oEllipticalArc.MinorRadius * 2
        Case kSketchLineObject
            Dim oLine As SketchLine
            Set oLine = oSketchEnt
            Debug.Print "Sketch Line selected."
            Debug.Print "  Start Point: " & oLine.StartSketchPoint.Geometry.X & _
                        ", " & oLine.StartSketchPoint.Geometry.Y
            Debug.Print "  End Point: " & oLine.EndSketchPoint.Geometry.X & _
                        ", " & oLine.EndSketchPoint.Geometry.Y
            Debug.Print "  Length: " & oLine.Length
            Debug.Print "  Centerline: " & oLine.Centerline

            ' Toggle the centerline property.
            If oLine.Centerline Then
                oLine.Centerline = False
            Else
                oLine.Centerline = True
            End If
        Case kSketchPointObject
            Dim oPoint As SketchPoint
            Set oPoint = oSketchEnt
            Debug.Print "Sketch Point selected."
            Debug.Print "  Position: " & oPoint.Geometry.X & _
                        ", " & oPoint.Geometry.Y
            Debug.Print "  Hole Center: " & oPoint.HoleCenter

            ' Toggle the hole center property.
            If oPoint.HoleCenter Then
                oPoint.HoleCenter = False
            Else
                oPoint.HoleCenter = True
            End If
        Case kSketchSplineObject
            Dim oSpline As SketchSpline
            Set oSpline = oSketchEnt
            Debug.Print "Sketch Spline selected."
            Debug.Print "  Length: " & oSpline.Length
            Debug.Print "  Closed: " & oSpline.Closed
            Debug.Print "  Fit Points (" & oSpline.FitPointCount & ")"
            Dim i As Long
            For i = 1 To oSpline.FitPointCount
                Debug.Print "    Fit Point " & i & ": " & oSpline.FitPoint(i).Geometry.X & _
                            ", " & oSpline.FitPoint(i).Geometry.Y
            Next
    End Select

    ' Call the generic sketch entity methods.
    Debug.Print "  Constraint count: " & oSketchEnt.Constraints.Count
    Debug.Print "  Construction: " & oSketchEnt.Construction
    Debug.Print "  Range: (" & oSketchEnt.RangeBox.MinPoint.X & ", " & _
                            oSketchEnt.RangeBox.MinPoint.Y & ") - (" & _
                            oSketchEnt.RangeBox.MaxPoint.X & ", " & _
                            oSketchEnt.RangeBox.MaxPoint.Y & ")"
    Debug.Print "  Reference: " & oSketchEnt.Reference
    If oSketchEnt.Reference Then
        Debug.Print "  Referenced Entity: " & TypeName(oSketchEnt.ReferencedEntity)
    End If
End Sub
```

To use this sample, select a sketch entity that you want to query. Run the program and it will display information about the selected entity in the immediate window.

Copy Code

```
    ' Get the first item in the select set.  This assumes
    ' something is selected and that it's a sketch entity.
    Dim oSketchEnt As SketchEntity
    oSketchEnt = ThisApplication.ActiveDocument.SelectSet.Item(1)

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Display type specific information.
    Select Case oSketchEnt.Type
        Case kSketchArcObject
            Dim oArc As SketchArc
            oArc = oSketchEnt
            Logger.Info("Sketch Arc selected.")

            Logger.Info("  Center Point: " & oArc.CenterSketchPoint.Geometry.X & _
                        ", " & oArc.CenterSketchPoint.Geometry.Y)
            Logger.Info("  Start Point: " & oArc.StartSketchPoint.Geometry.X & _
                        ", " & oArc.StartSketchPoint.Geometry.Y)
            Logger.Info("  End Point: " & oArc.EndSketchPoint.Geometry.X & _
                        ", " & oArc.EndSketchPoint.Geometry.Y)
            Logger.Info("  Start Angle: " & oArc.StartAngle)

            Logger.Info("  Sweep Angle: " & oArc.SweepAngle)

            Logger.Info("  Radius: " & oArc.Radius)

            Logger.Info("  Length: " & oArc.Length)

        Case kSketchCircleObject
            Dim oCircle As SketchCircle
```

```
    oCircle = oSketchEnt
    Logger.Info("Sketch Circle selected.")

    Logger.Info("  Center Point: " & oCircle.CenterSketchPoint.Geometry.X & _
                ", " & oCircle.CenterSketchPoint.Geometry.Y)
    Logger.Info("  Radius: " & oCircle.Radius)

    Logger.Info("  Area: " & oCircle.Area)


    ' Change the radius
    oCircle.Radius = oCircle.Radius * 1.5
Case kSketchEllipseObject
    Dim oEllipse As SketchEllipse
    oEllipse = oSketchEnt
    Logger.Info("Sketch Ellipse selected.")

    Logger.Info("  Center Point: " & oEllipse.CenterSketchPoint.Geometry.X & _
                ", " & oEllipse.CenterSketchPoint.Geometry.Y)
    Logger.Info("  Major Axis Vecotr: " & oEllipse.MajorAxisVector.X & _
                ", " & oEllipse.MajorAxisVector.Y)
    Logger.Info("  Major Radius: " & oEllipse.MajorRadius)

    Logger.Info("  Minor Radius: " & oEllipse.MinorRadius)

    Logger.Info("  Area: " & oEllipse.Area)


    ' Modify the ellipse
    oEllipse.MajorAxisVector = oTransGeom.CreateUnitVector2d(1, 1)
    oEllipse.MajorRadius = oEllipse.MajorRadius / 2
    oEllipse.MinorRadius = oEllipse.MinorRadius * 2
Case kSketchEllipticalArcObject
    Dim oEllipticalArc As SketchEllipticalArc
    oEllipticalArc = oSketchEnt
    Logger.Info("Sketch Elliptical Arc selected.")

    Logger.Info("  Center Point: " & oEllipticalArc.CenterSketchPoint.Geometry.X & _
                ", " & oEllipticalArc.CenterSketchPoint.Geometry.Y)
    Logger.Info("  Major Axis Vector: " & oEllipticalArc.MajorAxisVector.X & _
                ", " & oEllipticalArc.MajorAxisVector.Y)
    Logger.Info("  Major Radius: " & oEllipticalArc.MajorRadius)

    Logger.Info("  Minor Radius: " & oEllipticalArc.MinorRadius)

    Logger.Info("  Start Angle: " & oEllipticalArc.StartAngle)

    Logger.Info("  Sweep Angle: " & oEllipticalArc.SweepAngle)

    Logger.Info("  Length: " & oEllipticalArc.Length)


    ' Modify the elliptical arc.
    oEllipticalArc.MajorAxisVector = oTransGeom.CreateUnitVector2d(1, 1)
    oEllipticalArc.MajorRadius = oEllipticalArc.MajorRadius / 2
    oEllipticalArc.MinorRadius = oEllipticalArc.MinorRadius * 2
Case kSketchLineObject
    Dim oLine As SketchLine
    oLine = oSketchEnt
    Logger.Info("Sketch Line selected.")

    Logger.Info("  Start Point: " & oLine.StartSketchPoint.Geometry.X & _
                ", " & oLine.StartSketchPoint.Geometry.Y)
    Logger.Info("  End Point: " & oLine.EndSketchPoint.Geometry.X & _
                ", " & oLine.EndSketchPoint.Geometry.Y)
    Logger.Info("  Length: " & oLine.Length)

    Logger.Info("  Centerline: " & oLine.Centerline)


    ' Toggle the centerline property.
    If oLine.Centerline Then
        oLine.Centerline = False
    Else
        oLine.Centerline = True
    End If
Case kSketchPointObject
    Dim oPoint As SketchPoint
    oPoint = oSketchEnt
    Logger.Info("Sketch Point selected.")

    Logger.Info("  Position: " & oPoint.Geometry.X & _
                ", " & oPoint.Geometry.Y)
    Logger.Info("  Hole Center: " & oPoint.HoleCenter)


    ' Toggle the hole center property.
    If oPoint.HoleCenter Then
        oPoint.HoleCenter = False
    Else
        oPoint.HoleCenter = True
    End If
Case kSketchSplineObject
    Dim oSpline As SketchSpline
    oSpline = oSketchEnt
    Logger.Info("Sketch Spline selected.")

    Logger.Info("  Length: " & oSpline.Length)

    Logger.Info("  Closed: " & oSpline.Closed)

    Logger.Info("  Fit Points (" & oSpline.FitPointCount & ")")

    Dim i As Long
    For i = 1 To oSpline.FitPointCount
        Logger.Info("    Fit Point " & i & ": " & oSpline.FitPoint(i).Geometry.X & _
                    ", " & oSpline.FitPoint(i).Geometry.Y)
```

```
        Next
    End Select

    ' Call the generic sketch entity methods.
    Logger.Info("  Constraint count: " & oSketchEnt.Constraints.Count)

    Logger.Info("  Construction: " & oSketchEnt.Construction)

    Logger.Info("  Range: (" & oSketchEnt.RangeBox.MinPoint.X & ", " & _
                          oSketchEnt.RangeBox.MinPoint.Y & ") - (" & _
                          oSketchEnt.RangeBox.MaxPoint.X & ", " & _
                          oSketchEnt.RangeBox.MaxPoint.Y & ")")
    Logger.Info("  Reference: " & oSketchEnt.Reference)

    If oSketchEnt.Reference Then
        Logger.Info("  Referenced Entity: " & TypeName(oSketchEnt.ReferencedEntity))

    End If
```

Spline - create NURBS

Description

This sample demonstrates the creation of a sketch spline using a geometry definition (a NURB). The API also supports creation of 3D sketch splines in a similar way.

Code Samples

- VBA
- iLogic

Have a part document open and run the following sample.

Copy Code

```
Public Sub SplineByDefinition()
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create the collection that will contain the fit points for the regular spline.
    Dim oFitPoints As ObjectCollection
    Set oFitPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Define the points to fit the spline through. The
    ' points are at (0,0), (2,2), (4,0), (6,4).
    Dim oPoints(1 To 4) As Point2d
    Set oPoints(1) = oTransGeom.CreatePoint2d(0, 0)
    oFitPoints.Add oPoints(1)

    Set oPoints(2) = oTransGeom.CreatePoint2d(2, 2)
    oFitPoints.Add oPoints(2)

    Set oPoints(3) = oTransGeom.CreatePoint2d(4, 0)
    oFitPoints.Add oPoints(3)

    Set oPoints(4) = oTransGeom.CreatePoint2d(6, 4)
    oFitPoints.Add oPoints(4)

    'Create the first sketch on the X-Y plane
    Dim oSketch1 As PlanarSketch
    Set oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Create the spline.
    Dim oSpline As SketchSpline
    Set oSpline = oSketch1.SketchSplines.Add(oFitPoints)

    ' Create a work plane parallel to X-Y plane.
    Dim oWorkPlane As WorkPlane
    Set oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes(3), 2)

    'Create the second sketch.
    Dim oSketch2 As PlanarSketch
    Set oSketch2 = oCompDef.Sketches.Add(oWorkPlane)

    ' Set a reference to the geometry of the spline on the first sketch
    Dim oBSplineCurve2d As BSplineCurve2d
    Set oBSplineCurve2d = oSpline.Geometry

    ' Create a fixed spline on the second sketch based
    ' on the geometry of the spline on the first sketch.
    Dim oFixedSpline As SketchFixedSpline
    Set oFixedSpline = oSketch2.SketchFixedSplines.Add(oBSplineCurve2d)
End Sub
```

Have a part document open and run the following sample.

Copy Code

```
    ' Set a reference to the part component definition.
    ' This assumes that a part document is active.
    Dim oCompDef As PartComponentDefinition
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' Set a reference to the transient geometry collection.
```

```
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create the collection that will contain the fit points for the regular spline.
    Dim oFitPoints As ObjectCollection
    oFitPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Define the points to fit the spline through. The
    ' points are at (0,0), (2,2), (4,0), (6,4).
    Dim oPoints(0 To 3) As Point2d
    oPoints(0) = oTransGeom.CreatePoint2d(0, 0)
    oFitPoints.Add(oPoints(0))

    oPoints(1) = oTransGeom.CreatePoint2d(2, 2)
    oFitPoints.Add(oPoints(1))

    oPoints(2) = oTransGeom.CreatePoint2d(4, 0)
    oFitPoints.Add(oPoints(2))

    oPoints(3) = oTransGeom.CreatePoint2d(6, 4)
    oFitPoints.Add(oPoints(3))

    'Create the first sketch on the X-Y plane
    Dim oSketch1 As PlanarSketch
    oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Create the spline.
    Dim oSpline As SketchSpline
    oSpline = oSketch1.SketchSplines.Add(oFitPoints)

    ' Create a work plane parallel to X-Y plane.
    Dim oWorkPlane As WorkPlane
    oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes(3), 2)

    'Create the second sketch.
    Dim oSketch2 As PlanarSketch
    oSketch2 = oCompDef.Sketches.Add(oWorkPlane)

    ' Set a reference to the geometry of the spline on the first sketch
    Dim oBSplineCurve2d As BSplineCurve2d
    oBSplineCurve2d = oSpline.Geometry

    ' Create a fixed spline on the second sketch based
    ' on the geometry of the spline on the first sketch.
    Dim oFixedSpline As SketchFixedSpline
    oFixedSpline = oSketch2.SketchFixedSplines.Add(oBSplineCurve2d)
```

Create slots in sketch.

Description

This sample demonstrates several new methods to create sketch entities that represent slots. These are the equivalent to new sketch commands that were added in Inventor 2014.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSlots()
    ' Create a new part.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                  ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Create a 2D sketch on the X-Y plane.
    Dim sketch As PlanarSketch
    Set sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    Dim pi As Double
    pi = Atn(1) * 4

    Dim results As SketchEntitiesEnumerator
    Set results = sketch.AddArcSlotByCenterPointArc(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(6, 0), pi / 2, 2)

    Set results = sketch.AddArcSlotByThreePointArc(tg.CreatePoint2d(15, 0), tg.CreatePoint2d(12.5, 2), _
                                                   tg.CreatePoint2d(10, 0), 2)

    Set results = sketch.AddStraightSlotByCenterToCenter(tg.CreatePoint2d(0, 10), tg.CreatePoint2d(6, 10), 2)

    Set results = sketch.AddStraightSlotByOverall(tg.CreatePoint2d(10, 10), tg.CreatePoint2d(16, 10), 2)

    ThisApplication.ActiveView.Fit
End Sub
```

Copy Code

```
    ' Create a new part.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
```

```
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition

    ' Create a 2D sketch on the X-Y plane.
    Dim sketch As PlanarSketch
    sketch = partDef.Sketches.Add(partDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    Dim pi As Double
    pi = Math.Atan(1) * 4

    Dim results As SketchEntitiesEnumerator
    results = sketch.AddArcSlotByCenterPointArc(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(6, 0), pi / 2, 2)

    results = sketch.AddArcSlotByThreePointArc(tg.CreatePoint2d(15, 0), tg.CreatePoint2d(12.5, 2), _
                                               tg.CreatePoint2d(10, 0), 2)

    results = sketch.AddStraightSlotByCenterToCenter(tg.CreatePoint2d(0, 10), tg.CreatePoint2d(6, 10), 2)

    results = sketch.AddStraightSlotByOverall(tg.CreatePoint2d(10, 10), tg.CreatePoint2d(16, 10), 2)

    ThisApplication.ActiveView.Fit
```

Sketch Spline

Description

This sample demonstrates creating and manipulating a sketch spline.

Code Samples

- VBA
- iLogic

To use this sample a sketch must be active.

Copy Code

```
Public Sub DrawSketchSpline()
    ' Check to make sure a sketch is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox "A sketch must be active."
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create the collection that will contain the fit points for the spline.
    Dim oFitPoints As ObjectCollection
    Set oFitPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Define the points to fit the spline through.  In this example, transient
    ' points are used.  They could also be sketch points and then the spline
    ' will automatically be constrained to fit through the sketch point.  The
    ' points are at (0,0), (2,2), (4,0), (6,4), (7,-1).
    Dim oPoints(1 To 5) As Point2d
    Set oPoints(1) = oTransGeom.CreatePoint2d(0, 0)
    oFitPoints.Add oPoints(1)

    Set oPoints(2) = oTransGeom.CreatePoint2d(2, 2)
    oFitPoints.Add oPoints(2)

    Set oPoints(3) = oTransGeom.CreatePoint2d(4, 0)
    oFitPoints.Add oPoints(3)

    Set oPoints(4) = oTransGeom.CreatePoint2d(6, 4)
    oFitPoints.Add oPoints(4)

    Set oPoints(5) = oTransGeom.CreatePoint2d(7, -1)
    oFitPoints.Add oPoints(5)

    ' Create the spline.
    Dim oSpline As SketchSpline
    Set oSpline = oSketch.SketchSplines.Add(oFitPoints)

    ' Change the curve to be closed.
    oSpline.Closed = True

    ' Add a ground constraint to the third fit point.
    Call oSketch.GeometricConstraints.AddGround(oSpline.FitPoint(3))

    ' Add an additional fit point.
    Dim oNewPoint As Point2d
    Set oNewPoint = oTransGeom.CreatePoint2d(8, 8)
    Call oSpline.InsertFitPoint(oNewPoint, 5, True)

    ' Reposition the second fit point.
    Call oSpline.FitPoint(2).MoveTo(oTransGeom.CreatePoint2d(2, 3))

    ' Delete a fit point by deleting the underlying sketch point.
```

```
    oSpline.FitPoint(2).Delete
End Sub
```

To use this sample a sketch must be active.

Copy Code

```
    ' Check to make sure a sketch is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is PlanarSketch Then
        MsgBox("A sketch must be active.")
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As PlanarSketch
    oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create the collection that will contain the fit points for the spline.
    Dim oFitPoints As ObjectCollection
    oFitPoints = ThisApplication.TransientObjects.CreateObjectCollection

    ' Define the points to fit the spline through.  In this example, transient
    ' points are used.  They could also be sketch points and then the spline
    ' will automatically be constrained to fit through the sketch point.  The
    ' points are at (0,0), (2,2), (4,0), (6,4), (7,-1).
    Dim oPoints(0 To 4) As Point2d
    oPoints(0) = oTransGeom.CreatePoint2d(0, 0)
    oFitPoints.Add(oPoints(0))

    oPoints(1) = oTransGeom.CreatePoint2d(2, 2)
    oFitPoints.Add(oPoints(1))

    oPoints(2) = oTransGeom.CreatePoint2d(4, 0)
    oFitPoints.Add(oPoints(2))

    oPoints(3) = oTransGeom.CreatePoint2d(6, 4)
    oFitPoints.Add(oPoints(3))

    oPoints(4) = oTransGeom.CreatePoint2d(7, -1)
    oFitPoints.Add(oPoints(4))

    ' Create the spline.
    Dim oSpline As SketchSpline
    oSpline = oSketch.SketchSplines.Add(oFitPoints)

    ' Change the curve to be closed.
    oSpline.Closed = True

    ' Add a ground constraint to the third fit point.
    Call oSketch.GeometricConstraints.AddGround(oSpline.FitPoint(3))

    ' Add an additional fit point.
    Dim oNewPoint As Point2d
    oNewPoint = oTransGeom.CreatePoint2d(8, 8)
    Call oSpline.InsertFitPoint(oNewPoint, 5, True)

    ' Reposition the second fit point.
    Call oSpline.FitPoint(2).MoveTo(oTransGeom.CreatePoint2d(2, 3))

    ' Delete a fit point by deleting the underlying sketch point.
    oSpline.FitPoint(2).Delete
```

Sketch Text Add

Description

This sample illustrates creating text in a sketch.

Code Samples

- VBA
- iLogic

Before running this sample, open a drawing document.

Copy Code

```
Public Sub SketchTextAdd()
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    Set oDrawDoc = ThisApplication.ActiveDocument

    ' Create a new sketch on the active sheet.
    Dim oSketch As DrawingSketch
    Set oSketch = oDrawDoc.ActiveSheet.Sketches.Add

    ' Open the sketch for edit so the text boxes can be created.
    ' This is only required for drawing sketches, not part.
    oSketch.Edit

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create text with simple string as input.  Since this doesn't use
```

```
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "Drawing Notes"
    Dim oTextBox As TextBox
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 18), sText)

    ' Create text using various overrides.
    sText = "Notice: All holes larger than 0.500 n are to be lubricated."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 16), sText)

    ' Create a set of notes that are numbered and aligned along the left.
    Dim dYCoord As Double
    dYCoord = 14
    Dim dYOffset As Double
    Dim oStyle As TextStyle
    Set oStyle = oSketch.TextBoxes.Item(1).Style
    dYOffset = oStyle.FontSize * 1.5

    ' Simple single line text.
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "1.")
    sText = "This is note 1."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Two line text.  The two lines are defined using the  tag within the text string.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "2.")
    sText = "This is note 2,  which contains two lines."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Single line of text.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "3.")
    sText = "This is note 3."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Three lines of text.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "4.")
    sText = "This is note 4,  which contains  several lines."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' The number of this line of text will have a triangle around it.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "5")

    ' Draw a triangle that fits around a text box.
    Dim Points(1 To 3) As Point2d
    Call CalculateTriangle(oTextBox, Points)

    Dim oLines(1 To 3) As SketchLine
    Set oLines(1) = oSketch.SketchLines.AddByTwoPoints(Points(1), Points(2))
    Set oLines(2) = oSketch.SketchLines.AddByTwoPoints(oLines(1).EndSketchPoint, Points(3))
    Set oLines(3) = oSketch.SketchLines.AddByTwoPoints(oLines(2).EndSketchPoint, oLines(1).StartSketchPoint)

    sText = "Here is the last and final line of text."
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Exit the sketch from the edit environment.
    oSketch.ExitEdit
End Sub

' Function to calcuate the three points of a triangle that fits tightly around the input text box.
Private Sub CalculateTriangle(TextBox As TextBox, Points() As Point2d)
    ' Get the top-left corner of the text box.
    Dim dLeft As Double
    Dim dTop As Double
    Call GetCorner(TextBox, dLeft, dTop)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Determine the top point of the triangle.
    Set Points(1) = oTG.CreatePoint2d(dLeft + (TextBox.Width / 2), dTop + ((TextBox.Width / 2) * Tan(1.0471975511966)))

    ' Determine the bottom-left corner point of the triangle.
    Dim dY As Double
    dY = (Points(1).Y - dTop) + (1.125 * TextBox.height)
    Set Points(2) = oTG.CreatePoint2d(Points(1).X - (dY / Tan(1.0471975511966)), Points(1).Y - dY)

    ' Determine the bottom right corner point of the triangle
    Set Points(3) = oTG.CreatePoint2d(Points(1).X + (dY / Tan(1.0471975511966)), Points(2).Y)
End Sub

' Function to determine the top left corner of the input text box.
Private Sub GetCorner(TextBox As TextBox, Left As Double, Top As Double)
    ' Determine the top left corner of the text box by accounting
    ' for the justifications.
    Select Case TextBox.HorizontalJustification
        Case kAlignTextLeft
            Left = TextBox.Origin.X
        Case kAlignTextCenter
            Left = TextBox.Origin.X - (TextBox.Width / 2)
        Case kAlignTextRight
            Left = TextBox.Origin.X - TextBox.Width
    End Select

    Select Case TextBox.VerticalJustification
        Case kAlignTextUpper
            Top = TextBox.Origin.Y
        Case kAlignTextMiddle
            Top = TextBox.Origin.Y + (TextBox.height / 2)
        Case kAlignTextLower
            Top = TextBox.Origin.Y + TextBox.height
    End Select
End Sub
```

Before running this sample, open a drawing document.

Copy Code

```
Sub Main
    ' Set a reference to the drawing document.
    ' This assumes a drawing document is active.
    Dim oDrawDoc As DrawingDocument
    oDrawDoc = ThisApplication.ActiveDocument

    ' Create a new sketch on the active sheet.
    Dim oSketch As DrawingSketch
    oSketch = oDrawDoc.ActiveSheet.Sketches.Add()

    ' Open the sketch for edit so the text boxes can be created.
    ' This is only required for drawing sketches, not part.
    oSketch.Edit

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create text with simple string as input.  Since this doesn't use
    ' any text overrides, it will default to the active text style.
    Dim sText As String
    sText = "Drawing Notes"
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 18), sText)

    ' Create text using various overrides.
    sText = "Notice: All holes larger than 0.500 n are to be lubricated."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 16), sText)

    ' Create a set of notes that are numbered and aligned along the left.
    Dim dYCoord As Double
    dYCoord = 14
    Dim dYOffset As Double
    Dim oStyle As TextStyle
    oStyle = oSketch.TextBoxes.Item(1).Style
    dYOffset = oStyle.FontSize * 1.5

    ' Simple single line text.
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "1.")
    sText = "This is note 1."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Two line text.  The two lines are defined using the  tag within the text string.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "2.")
    sText = "This is note 2,  which contains two lines."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Single line of text.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "3.")
    sText = "This is note 3."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Three lines of text.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "4.")
    sText = "This is note 4,  which contains  several lines."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' The number of this line of text will have a triangle around it.
    dYCoord = dYCoord - (oTextBox.FittedTextHeight + 0.5)
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, dYCoord), "5")

    ' Draw a triangle that fits around a text box.
    Dim Points(0 To 2) As Point2d
    Call CalculateTriangle(oTextBox, Points)

    Dim oLines(0 To 2) As SketchLine
    oLines(0) = oSketch.SketchLines.AddByTwoPoints(Points(0), Points(1))
    oLines(1) = oSketch.SketchLines.AddByTwoPoints(oLines(0).EndSketchPoint, Points(2))
    oLines(2) = oSketch.SketchLines.AddByTwoPoints(oLines(1).EndSketchPoint, oLines(0).StartSketchPoint)

    sText = "Here is the last and final line of text."
    oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(4, dYCoord), sText)

    ' Exit the sketch from the edit environment.
    oSketch.ExitEdit
End Sub

' Function to calcuate the three points of a triangle that fits tightly around the input text box.
Private Sub CalculateTriangle(TextBox As TextBox, Points() As Point2d)
    ' Get the top-left corner of the text box.
    Dim dLeft As Double
    Dim dTop As Double
    Call GetCorner(TextBox, dLeft, dTop)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Determine the top point of the triangle.
    Points(0) = oTG.CreatePoint2d(dLeft + (TextBox.Width / 2), dTop + ((TextBox.Width / 2) * Tan(1.0471975511966)))

    ' Determine the bottom-left corner point of the triangle.
    Dim dY As Double
    dY = (Points(0).Y - dTop) + (1.125 * TextBox.Height)
    Points(1) = oTG.CreatePoint2d(Points(0).X - (dY / Tan(1.0471975511966)), Points(0).Y - dY)

    ' Determine the bottom right corner point of the triangle
    Points(2) = oTG.CreatePoint2d(Points(0).X + (dY / Tan(1.0471975511966)), Points(1).Y)
End Sub

' Function to determine the top left corner of the input text box.
Private Sub GetCorner(TextBox As TextBox, Left As Double, Top As Double)
    ' Determine the top left corner of the text box by accounting
    ' for the justifications.
```

```
    Select Case TextBox.HorizontalJustification
        Case kAlignTextLeft
            Left = TextBox.Origin.X
        Case kAlignTextCenter
            Left = TextBox.Origin.X - (TextBox.Width / 2)
        Case kAlignTextRight
            Left = TextBox.Origin.X - TextBox.Width
    End Select

    Select Case TextBox.VerticalJustification
        Case kAlignTextUpper
            Top = TextBox.Origin.Y
        Case kAlignTextMiddle
            Top = TextBox.Origin.Y + (TextBox.height / 2)
        Case kAlignTextLower
            Top = TextBox.Origin.Y + TextBox.height
    End Select
End Sub
```

Create a 3D sketch dimension

Description

This sample demonstrates the creation of a 3d sketch line and a dimension between the start and the end points of the line.

Code Samples

- VBA
- iLogic

To run this sample you must have a 3D sketch active.

Copy Code

```
Public Sub Create3DSketchDimension()
    ' Check to make sure a 3d sketch is open.
    If Not Typeof ThisApplication.ActiveEditObject Is Sketch3D Then
        MsgBox "A 3d sketch must be active."
        Exit Sub
    End If

    ' Set a reference to the active sketch.
    Dim oSketch As Sketch3D
    Set oSketch = ThisApplication.ActiveEditObject

    ' Set a reference to the transient geometry collection.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a sketch line
    Dim oSketchLine As SketchLine3D
    Set oSketchLine = oSketch.SketchLines3D.AddByTwoPoints(oTransGeom.CreatePoint(5, 5, 0), oTransGeom.CreatePoint(10, 10, 20))

    ' Create an object collection and add the end points.
    Dim oColl As ObjectCollection
    Set oColl = ThisApplication.TransientObjects.CreateObjectCollection

    oColl.Add oSketchLine.StartSketchPoint
    oColl.Add oSketchLine.EndSketchPoint

    ' The text point for the dimension needs to be on a plane
    ' that Inventor internally computes based on the sketch
    ' entities being dimensioned. Get this plane.
    Dim oDimPlane As Plane
    Set oDimPlane = oSketch.DimensionConstraints3D.GetDimensionPlane(oColl)

    ' Get the normal of the plane
    Dim oPlaneNormal As UnitVector
    Set oPlaneNormal = oDimPlane.Normal

    ' Get the sketch line direction
    Dim oLineDir As UnitVector
    Set oLineDir = oSketchLine.Geometry.Direction

    ' Get the direction in which to offset the text point.
    ' (This should be perpendicular to the plane normal
    ' and the line direction).
    Dim oOffsetDir As UnitVector
    Set oOffsetDir = oPlaneNormal.CrossProduct(oLineDir)

    ' Get the midpoint of the sketch line
    Dim oMidPoint As Point
    Set oMidPoint = oSketchLine.Geometry.MidPoint

    ' Offset the midpoint from the line by a distance.
    Dim oOffsetPoint As Point
    Set oOffsetPoint = oMidPoint
    Call oOffsetPoint.TranslateBy(oOffsetDir.AsVector)

    ' Create the dimension.
    Dim oTwoPointDim As TwoPointDistanceDimConstraint3D
    Set oTwoPointDim = oSketch.DimensionConstraints3D.AddTwoPointDistance(oSketchLine.StartSketchPoint, oSketchLine.EndSketchPoint, oO

    ' Make sure that the dimension text was placed at the specified point.
    If oOffsetPoint.IsEqualTo(oTwoPointDim.TextPoint) Then
        MsgBox "Dimension text placed at specified point."
    End If
End Sub
```

To run this sample you must have a 3D sketch active.

Copy Code

```
' Check to make sure a 3d sketch is open.
If Not Typeof ThisApplication.ActiveEditObject Is Sketch3D Then
    MsgBox("A 3d sketch must be active.")
    Exit Sub
End If

' Set a reference to the active sketch.
Dim oSketch As Sketch3D
oSketch = ThisApplication.ActiveEditObject

' Set a reference to the transient geometry collection.
Dim oTransGeom As TransientGeometry
oTransGeom = ThisApplication.TransientGeometry

' Create a sketch line
Dim oSketchLine As SketchLine3D
oSketchLine = oSketch.SketchLines3D.AddByTwoPoints(oTransGeom.CreatePoint(5, 5, 0), oTransGeom.CreatePoint(10, 10, 20))

' Create an object collection and add the end points.
Dim oColl As ObjectCollection
oColl = ThisApplication.TransientObjects.CreateObjectCollection

oColl.Add(oSketchLine.StartSketchPoint)
oColl.Add(oSketchLine.EndSketchPoint)

' The text point for the dimension needs to be on a plane
' that Inventor internally computes based on the sketch
' entities being dimensioned. Get this plane.
Dim oDimPlane As Plane
oDimPlane = oSketch.DimensionConstraints3D.GetDimensionPlane(oColl)

' Get the normal of the plane
Dim oPlaneNormal As UnitVector
oPlaneNormal = oDimPlane.Normal

' Get the sketch line direction
Dim oLineDir As UnitVector
oLineDir = oSketchLine.Geometry.Direction

' Get the direction in which to offset the text point.
' (This should be perpendicular to the plane normal
' and the line direction).
Dim oOffsetDir As UnitVector
oOffsetDir = oPlaneNormal.CrossProduct(oLineDir)

' Get the midpoint of the sketch line
Dim oMidPoint As Point
oMidPoint = oSketchLine.Geometry.MidPoint

' Offset the midpoint from the line by a distance.
Dim oOffsetPoint As Point
oOffsetPoint = oMidPoint
Call oOffsetPoint.TranslateBy(oOffsetDir.AsVector)

' Create the dimension.
Dim oTwoPointDim As TwoPointDistanceDimConstraint3D
oTwoPointDim = oSketch.DimensionConstraints3D.AddTwoPointDistance(oSketchLine.StartSketchPoint, oSketchLine.EndSketchPoint, oOffse

' Make sure that the dimension text was placed at the specified point.
If oOffsetPoint.IsEqualTo(oTwoPointDim.TextPoint) Then
    MsgBox("Dimension text placed at specified point.")
End If
```

OnFaceCurve creation

Description

This sample demonstrates how to create a OnFaceCurve in 3D sketch.

Code Samples

- VBA
- iLogic

Open a part document with a surface body in it, and then run below VBA code.

Copy Code

```
Sub OnFaceCurveSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oFace As Face
    Set oFace = oDoc.ComponentDefinition.SurfaceBodies(1).Faces(1)

    Dim oSk3D As Sketch3D
    Set oSk3D = oDoc.ComponentDefinition.Sketches3D.Add

    Dim oFaces As NameValueMap
    Dim oFitPoints As NameValueMap

    Set oFaces = ThisApplication.TransientObjects.CreateNameValueMap
    Set oFitPoints = ThisApplication.TransientObjects.CreateNameValueMap
```

```
    Dim i As Long, oTempFace As Face, oCol As ObjectCollection
    For i = 1 To oFace.FaceShell.Faces.Count
        Set oCol = ThisApplication.TransientObjects.CreateObjectCollection

        Set oTempFace = oFace.FaceShell.Faces(i)
        oCol.Add oTempFace.PointOnFace

        oFaces.Add "Face" & i, oTempFace
        oFitPoints.Add "Face" & i, oCol
    Next

    Dim oOnFaceCurve  As OnFaceCurve
    Set oOnFaceCurve = oSk3D.OnFaceCurves.Add(oFaces, oFitPoints)
End Sub
```

Open a part document with a surface body in it, and then run below VBA code.

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oFace As Face
    oFace = oDoc.ComponentDefinition.SurfaceBodies(1).Faces(1)

    Dim oSk3D As Sketch3D
    oSk3D = oDoc.ComponentDefinition.Sketches3D.Add

    Dim oFaces As NameValueMap
    Dim oFitPoints As NameValueMap

    oFaces = ThisApplication.TransientObjects.CreateNameValueMap
    oFitPoints = ThisApplication.TransientObjects.CreateNameValueMap

    Dim i As Long, oTempFace As Face, oCol As ObjectCollection
    For i = 1 To oFace.FaceShell.Faces.Count
        oCol = ThisApplication.TransientObjects.CreateObjectCollection

        oTempFace = oFace.FaceShell.Faces(i)
        oCol.Add(oTempFace.PointOnFace)

        oFaces.Add("Face" & i, oTempFace)
        oFitPoints.Add("Face" & i, oCol)
    Next

    Dim oOnFaceCurve  As OnFaceCurve
    oOnFaceCurve = oSk3D.OnFaceCurves.Add(oFaces, oFitPoints)
```

Delete Face, Boundary Patch and Stitch features

Description

Demonstrates creating Face, Boundary Patch and Stitch features.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub BoundaryPatchFeature()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion
    Dim oFrontFace As Face
```

```
    Set oFrontFace = oExtrude.StartFaces.Item(1)

    Dim oEdgeColl As EdgeCollection
    Set oEdgeColl = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Collect up the edges to create boundary patch.
    Dim oEdge As Edge
    For Each oEdge In oFrontFace.Edges
        oEdgeColl.Add oEdge
    Next

    ' Create a Delete Face feature to delete the top face.
    Dim oFaceColl As FaceCollection
    Set oFaceColl = ThisApplication.TransientObjects.CreateFaceCollection
    Call oFaceColl.Add(oFrontFace)

    Dim oDeleteFace As DeleteFaceFeature
    Set oDeleteFace = oCompDef.Features.DeleteFaceFeatures.Add(oFaceColl)

    Dim oBoundaryPatchDef As BoundaryPatchDefinition
    Set oBoundaryPatchDef = oCompDef.Features.BoundaryPatchFeatures.CreateBoundaryPatchDefinition

    ' Create a boundary patch definition based on the edges of the deleted face.
    Call oBoundaryPatchDef.BoundaryPatchLoops.Add(oEdgeColl)

    ' Set the conditions on each edge to be tangent.
    For Each oEdge In oEdgeColl
        Call oBoundaryPatchDef.BoundaryPatchLoops.Item(1).SetBoundaryCondition(oEdge, kTangentBoundaryPatchCondition)
    Next

    ' Create the boundary patch feature based on the definition.
    Dim oBoundaryPatch As BoundaryPatchFeature
    Set oBoundaryPatch = oCompDef.Features.BoundaryPatchFeatures.Add(oBoundaryPatchDef)

    ' Stitch the boundary patch surface to the original body.
    Dim oSurfaceToStitch As ObjectCollection
    Set oSurfaceToStitch = ThisApplication.TransientObjects.CreateObjectCollection

    Call oSurfaceToStitch.Add(oBoundaryPatch.SurfaceBodies.Item(1))
    Call oSurfaceToStitch.Add(oCompDef.SurfaceBodies.Item(1))

    Dim oStitch As KnitFeature
    Set oStitch = oCompDef.Features.KnitFeatures.Add(oSurfaceToStitch)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
               ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion
    Dim oFrontFace As Face
    oFrontFace = oExtrude.StartFaces.Item(1)

    Dim oEdgeColl As EdgeCollection
    oEdgeColl = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Collect up the edges to create boundary patch.
    Dim oEdge As Edge
    For Each oEdge In oFrontFace.Edges
        oEdgeColl.Add(oEdge)
    Next

    ' Create a Delete Face feature to delete the top face.
    Dim oFaceColl As FaceCollection
    oFaceColl = ThisApplication.TransientObjects.CreateFaceCollection
    Call oFaceColl.Add(oFrontFace)

    Dim oDeleteFace As DeleteFaceFeature
    oDeleteFace = oCompDef.Features.DeleteFaceFeatures.Add(oFaceColl)

    Dim oBoundaryPatchDef As BoundaryPatchDefinition
    oBoundaryPatchDef = oCompDef.Features.BoundaryPatchFeatures.CreateBoundaryPatchDefinition

    ' Create a boundary patch definition based on the edges of the deleted face.
    Call oBoundaryPatchDef.BoundaryPatchLoops.Add(oEdgeColl)
```

```
    ' Set the conditions on each edge to be tangent.
    For Each oEdge In oEdgeColl
        Call oBoundaryPatchDef.BoundaryPatchLoops.Item(1).SetBoundaryCondition(oEdge, kTangentBoundaryPatchCondition)
    Next

    ' Create the boundary patch feature based on the definition.
    Dim oBoundaryPatch As BoundaryPatchFeature
    oBoundaryPatch = oCompDef.Features.BoundaryPatchFeatures.Add(oBoundaryPatchDef)

    ' Stitch the boundary patch surface to the original body.
    Dim oSurfaceToStitch As ObjectCollection
    oSurfaceToStitch = ThisApplication.TransientObjects.CreateObjectCollection

    Call oSurfaceToStitch.Add(oBoundaryPatch.SurfaceBodies.Item(1))
    Call oSurfaceToStitch.Add(oCompDef.SurfaceBodies.Item(1))

    Dim oStitch As KnitFeature
    oStitch = oCompDef.Features.KnitFeatures.Add(oSurfaceToStitch)
```

SurfaceBody Copy

Description

This sample demonstrates copying a surface body from one part to another. This is equivalent to the Promote command, but the API is much more flexible. In order for the sample to be self-contained, it creates two parts on the fly that will be used to demonstrate copying a body from one part to another. When copying a body into a part, you provide the surface body and a matrix to define its position in the new part. This sample creates a matrix based on the position of these parts within an assembly.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CopyBodyFromPartToPart()
    ' The first portion of this program creates an assembly so that the
    ' copy body API can be demonstrated. Any assembly could potentially
    ' be used, but the sample is simpler if it's for a specific case.
    '
    ' This creates an assembly that contains two parts. An interesting
    ' aspect of the sample is that the assembly and parts only exist
    ' in memory and are not written to disk. Filenames are assigned,
    ' so that if you perform a Save they will be written to disk using
    ' the specified filenames.

    ' Create a new assembly.
    Dim oAsmDoc As AssemblyDocument
    Set oAsmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))

    ' Define the filename for the assembly. It won't be saved at this point, but
    ' this name will be used if the assembly is saved later by the user.
    oAsmDoc.FullFileName = "C:\Temp\CopyBodyTestAsm.iam"

    ' Create a new part, invisibly.
    Dim oPartDoc1 As PartDocument
    Set oPartDoc1 = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject), False)

    ' Define the filename for the part. It won't be saved at this point, but
    ' this name will be used if the part is saved later by the user.
    oPartDoc1.FullFileName = "C:\Temp\CopyBodyTestPart1.ipt"

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc1.ComponentDefinition

    ' Create an extrude feature.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(1))
    Call oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(0, 0), 2)
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(3, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a second extrude feature as a work surface.
    Set oSketch = oPartDoc1.ComponentDefinition.Sketches.Add( _
                    oPartDoc1.ComponentDefinition.WorkPlanes.Item(1))
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-3, -3), oTG.CreatePoint2d(3, 3))
    Set oProfile = oSketch.Profiles.AddForSolid

    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(1.5, kPositiveExtentDirection)
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Insert the occurrence into the assembly using a somewhat arbitrary position.
    Dim oMatrix As Matrix
    Set oMatrix = oTG.CreateMatrix
```

```
    Call oMatrix.Translation.AddVector(oTG.CreateVector(2, 3, 4))
    Call oMatrix.SetToRotation(0.5, oTG.CreateVector(1, 0, 0), oTG.CreatePoint(2, 3, 4))
    Dim oOcc1 As ComponentOccurrence
    Set oOcc1 = oAsmDoc.ComponentDefinition.Occurrences.AddByComponentDefinition( _
                                    oPartDoc1.ComponentDefinition, oMatrix)

    ' Create a second new part, invisibly.
    Dim oPartDoc2 As PartDocument
    Set oPartDoc2 = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject), False)

    ' Define the filename for the part. It won't be saved at this point, but
    ' this name will be used if the part is saved later by the user.
    oPartDoc2.FullFileName = "C:\Temp\CopyBodyTestPart2.ipt"

    ' Insert the second part into the assembly using a somewhat arbitrary position.
    Set oMatrix = oTG.CreateMatrix
    Call oMatrix.Translation.AddVector(oTG.CreateVector(-1, -1, -1))
    Call oMatrix.SetToRotation(0.5, oTG.CreateVector(1, 1, 0), oTG.CreatePoint(1, 1, 1))
    Dim oOcc2 As ComponentOccurrence
    Set oOcc2 = oAsmDoc.ComponentDefinition.Occurrences.AddByComponentDefinition( _
                                    oPartDoc2.ComponentDefinition, oMatrix)

    ' Get the surface body from the first part that represents the work surface.
    ' In this case we know there's only one work surface so this just gets the
    ' first work surface in the collection.
    Dim oWorkSurface As WorkSurface
    Set oWorkSurface = oPartDoc1.ComponentDefinition.WorkSurfaces.Item(1)
    Dim oBody As SurfaceBody
    Set oBody = oWorkSurface.SurfaceBodies.Item(1)

    ' Define the matrix to use in copying the surface body from one part to
    ' another. Any matrix can be used, but in this case I want the position
    ' of the body to be the same with respect to assembly space. Because
    ' the occurrences are in different positions within the assembly the matrix
    ' needs to take into account the transfrom from one occurrence to the other.

    ' Get the transform from the occurrence where the surface body currently exists.
    ' This defines the tranform of the surface body into assembly space.
    Set oMatrix = oOcc1.Transformation

    ' Get the matrix of the second occurrence and invert it.
    ' The inverse of the second occurrence's transform defines the tranform from
    ' assembly space into that occurrence's part space.
    Dim oMatrix2 As Matrix
    Set oMatrix2 = oOcc2.Transformation
    oMatrix2.Invert

    ' Combine these matrices.
    Call oMatrix.PreMultiplyBy(oMatrix2)

    ' Copy the surface body from the first part into the second. You shouldn't
    ' see anything graphically change because the new body is directly on top of
    ' the existing body, but it can be verified because the new body is in
    ' the second part.
    Call oPartDoc2.ComponentDefinition.Features.NonParametricBaseFeatures.Add(oBody, oMatrix)
End Sub
```

Copy Code

```
    ' The first portion of this program creates an assembly so that the
    ' copy body API can be demonstrated. Any assembly could potentially
    ' be used, but the sample is simpler if it's for a specific case.
    '
    ' This creates an assembly that contains two parts. An interesting
    ' aspect of the sample is that the assembly and parts only exist
    ' in memory and are not written to disk. Filenames are assigned,
    ' so that if you perform a Save they will be written to disk using
    ' the specified filenames.

    ' Create a new assembly.
    Dim oAsmDoc As AssemblyDocument
    oAsmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))

    ' Define the filename for the assembly. It won't be saved at this point, but
    ' this name will be used if the assembly is saved later by the user.
    oAsmDoc.FullFileName = "C:\Temp\CopyBodyTestAsm.iam"

    ' Create a new part, invisibly.
    Dim oPartDoc1 As PartDocument
    oPartDoc1 = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject), False)

    ' Define the filename for the part. It won't be saved at this point, but
    ' this name will be used if the part is saved later by the user.
    oPartDoc1.FullFileName = "C:\Temp\CopyBodyTestPart1.ipt"

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc1.ComponentDefinition

    ' Create an extrude feature.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(1))
    Call oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(0, 0), 2)
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(3, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
```

```
    ' Create a second extrude feature as a work surface.
    oSketch = oPartDoc1.ComponentDefinition.Sketches.Add( _
                    oPartDoc1.ComponentDefinition.WorkPlanes.Item(1))
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-3, -3), oTG.CreatePoint2d(3, 3))
    oProfile = oSketch.Profiles.AddForSolid

    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(1.5, kPositiveExtentDirection)
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Insert the occurrence into the assembly using a somewhat arbitrary position.
    Dim oMatrix As Matrix
    oMatrix = oTG.CreateMatrix
    Call oMatrix.Translation.AddVector(oTG.CreateVector(2, 3, 4))
    Call oMatrix.SetToRotation(0.5, oTG.CreateVector(1, 0, 0), oTG.CreatePoint(2, 3, 4))
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmDoc.ComponentDefinition.Occurrences.AddByComponentDefinition( _
                                    oPartDoc1.ComponentDefinition, oMatrix)

    ' Create a second new part, invisibly.
    Dim oPartDoc2 As PartDocument
    oPartDoc2 = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject), False)

    ' Define the filename for the part. It won't be saved at this point, but
    ' this name will be used if the part is saved later by the user.
    oPartDoc2.FullFileName = "C:\Temp\CopyBodyTestPart2.ipt"

    ' Insert the second part into the assembly using a somewhat arbitrary position.
    oMatrix = oTG.CreateMatrix
    Call oMatrix.Translation.AddVector(oTG.CreateVector(-1, -1, -1))
    Call oMatrix.SetToRotation(0.5, oTG.CreateVector(1, 1, 0), oTG.CreatePoint(1, 1, 1))
    Dim oOcc2 As ComponentOccurrence
    oOcc2 = oAsmDoc.ComponentDefinition.Occurrences.AddByComponentDefinition( _
                                    oPartDoc2.ComponentDefinition, oMatrix)

    ' Get the surface body from the first part that represents the work surface.
    ' In this case we know there's only one work surface so this just gets the
    ' first work surface in the collection.
    Dim oWorkSurface As WorkSurface
    oWorkSurface = oPartDoc1.ComponentDefinition.WorkSurfaces.Item(1)
    Dim oBody As SurfaceBody
    oBody = oWorkSurface.SurfaceBodies.Item(1)

    ' Define the matrix to use in copying the surface body from one part to
    ' another. Any matrix can be used, but in this case I want the position
    ' of the body to be the same with respect to assembly space. Because
    ' the occurrences are in different positions within the assembly the matrix
    ' needs to take into account the transfrom from one occurrence to the other.

    ' Get the transform from the occurrence where the surface body currently exists.
    ' This defines the tranform of the surface body into assembly space.
    oMatrix = oOcc1.Transformation

    ' Get the matrix of the second occurrence and invert it.
    ' The inverse of the second occurrence's transform defines the tranform from
    ' assembly space into that occurrence's part space.
    Dim oMatrix2 As Matrix
    oMatrix2 = oOcc2.Transformation
    oMatrix2.Invert

    ' Combine these matrices.
    Call oMatrix.PreMultiplyBy(oMatrix2)

    ' Copy the surface body from the first part into the second. You shouldn't
    ' see anything graphically change because the new body is directly on top of
    ' the existing body, but it can be verified because the new body is in
    ' the second part.
    Call oPartDoc2.ComponentDefinition.Features.NonParametricBaseFeatures.Add(oBody, oMatrix)
```

Create Pattern Feature with PatternBoundarySetting Sample

Description

This sample demonstrates how to create a rectangular pattern feature with boundary settings.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a rectangular pattern feature with boundary settings.

Copy Code

```
Public Sub CreatePatternBoundarySettingSample()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
```

```
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a rectangle.
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(-10, -10), _
                              oTransGeom.CreatePoint2d(10, 10))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    Set oFrontFace = oExtrude.StartFaces.Item(1)

    ' Create a new sketch on this face.
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                  oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    Set oCorner = oTransGeom.CreatePoint2d(-9, 9)

    Dim oTriangle As SketchEntitiesEnumerator
    Set oTriangle = oSketch.SketchLines.AddAsPolygon(3, oCorner, oTransGeom.CreatePoint2d(-9.5, 9.5), True)

    ' Create a profile.
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a cut feature.
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                  oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTransGeom.CreatePoint2d(-6.5, 8), oTransGeom.CreatePoint2d(8.5, -8))

    ' Use the cut feature as parent features for pattern feature.
    Dim oParentFeatures As ObjectCollection
    Set oParentFeatures = ThisApplication.TransientObjects.CreateObjectCollection
    oParentFeatures.Add oExtrude

    Dim oRectPatternDef As RectangularPatternFeatureDefinition
    Set oRectPatternDef = oCompDef.Features.RectangularPatternFeatures.CreateDefinition(oParentFeatures, oFrontFace.Edges(6), False, 6
    oRectPatternDef.YDirectionEntity = oFrontFace.Edges(7)
    oRectPatternDef.YCount = 6
    oRectPatternDef.YSpacing = "3 cm"
    oRectPatternDef.NaturalYDirection = True

    ' Set the boundary settings for the pattern feature.
    Set oProfile = oSketch.Profiles.AddForSolid
    Dim oPatternBoundarySetting As PatternBoundarySetting
    Set oPatternBoundarySetting = oRectPatternDef.BoundarySetting
    Call oPatternBoundarySetting.SetBoundarySettingData(oProfile, kCentroidsInclusionType)

    ' Create rectangular pattern feature with boundary settings.
    Dim oRectPatternFeature As RectangularPatternFeature
    Set oRectPatternFeature = oCompDef.Features.RectangularPatternFeatures.AddByDefinition(oRectPatternDef)
End Sub
```

This sample demonstrates how to create a rectangular pattern feature with boundary settings.

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a rectangle.
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(-10, -10), _
                              oTransGeom.CreatePoint2d(10, 10))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
```

```
    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude.StartFaces.Item(1)

    ' Create a new sketch on this face.
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    oCorner = oTransGeom.CreatePoint2d(-9, 9)

    Dim oTriangle As SketchEntitiesEnumerator
    oTriangle = oSketch.SketchLines.AddAsPolygon(3, oCorner, oTransGeom.CreatePoint2d(-9.5, 9.5), True)

    ' Create a profile.
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a cut feature.
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTransGeom.CreatePoint2d(-6.5, 8), oTransGeom.CreatePoint2d(8.5, -8))

    ' Use the cut feature as parent features for pattern feature.
    Dim oParentFeatures As ObjectCollection
    oParentFeatures = ThisApplication.TransientObjects.CreateObjectCollection
    oParentFeatures.Add(oExtrude)

    Dim oRectPatternDef As RectangularPatternFeatureDefinition
    oRectPatternDef = oCompDef.Features.RectangularPatternFeatures.CreateDefinition(oParentFeatures, oFrontFace.Edges(6), False, 6, "4
    oRectPatternDef.YDirectionEntity = oFrontFace.Edges(7)
    oRectPatternDef.YCount = 6
    oRectPatternDef.YSpacing = "3 cm"
    oRectPatternDef.NaturalYDirection = True

    ' Set the boundary settings for the pattern feature.
    oProfile = oSketch.Profiles.AddForSolid
    Dim oPatternBoundarySetting As PatternBoundarySetting
    oPatternBoundarySetting = oRectPatternDef.BoundarySetting
    Call oPatternBoundarySetting.SetBoundarySettingData(oProfile, kCentroidsInclusionType)

    ' Create rectangular pattern feature with boundary settings.
    Dim oRectPatternFeature As RectangularPatternFeature
    oRectPatternFeature = oCompDef.Features.RectangularPatternFeatures.AddByDefinition(oRectPatternDef)
```

Add a decal feature

Description

This sample demonstrates the creation of a decal feature.

Code Samples

- VBA
- iLogic

Make sure that the path to the bmp file is valid before running the sample.

Copy Code

```
Public Sub DecalFeature()
    ' ***Change path to point to the desired bmp file.
    Dim strImagePath As String
    strImagePath = "C:\Temp\Test.bmp"

    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry
    Dim oCenter As Point2d
    Set oCenter = oTransGeom.CreatePoint2d(0, 0)

    ' Create a sketch circle
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius(oCenter, 1)

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 4 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
```

```
    Call oExtrudeDef.SetDistanceExtent(1, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a new sketch on the Y-Z work plane.
    Dim oDecalSketch As PlanarSketch
    Set oDecalSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(1))

    ' Create the placement point for the image.
    Dim oPoint As Point2d
    Set oPoint = oTransGeom.CreatePoint2d(0, 4)

    ' Add a sketch image
    Dim oSketchImage As SketchImage
    Set oSketchImage = oDecalSketch.SketchImages.Add(strImagePath, oPoint)

    ' Get the cylindrical face of the extrude
    Dim oFace As Face
    Set oFace = oExtrude.SideFaces.Item(1)

    ' Create a decal feature that wraps onto the cylindrical face.
    Dim oDecal As DecalFeature
    Set oDecal = oCompDef.Features.DecalFeatures.Add(oSketchImage, oFace, True)
End Sub
```

Make sure that the path to the bmp file is valid before running the sample.

Copy Code

```
    ' ***Change path to point to the desired bmp file.
    Dim strImagePath As String
    strImagePath = "C:\Temp\Test.bmp"

    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
               ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry
    Dim oCenter As Point2d
    oCenter = oTransGeom.CreatePoint2d(0, 0)

    ' Create a sketch circle
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius(oCenter, 1)

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 4 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a new sketch on the Y-Z work plane.
    Dim oDecalSketch As PlanarSketch
    oDecalSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(1))

    ' Create the placement point for the image.
    Dim oPoint As Point2d
    oPoint = oTransGeom.CreatePoint2d(0, 4)

    ' Add a sketch image
    Dim oSketchImage As SketchImage
    oSketchImage = oDecalSketch.SketchImages.Add(strImagePath, oPoint)

    ' Get the cylindrical face of the extrude
    Dim oFace As Face
    oFace = oExtrude.SideFaces.Item(1)

    ' Create a decal feature that wraps onto the cylindrical face.
    Dim oDecal As DecalFeature
    oDecal = oCompDef.Features.DecalFeatures.Add(oSketchImage, oFace, True)
```

Display feature information

Description

Displays information about all of the extrude features in the active document. A part document must be active when this is run.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub GetFeatureInfo()
    ' Get the active document assuming it is a part.
    Dim partDoc As PartDocument
    Set partDoc = ThisApplication.ActiveDocument

    ' Get the component definition.  This owns the part specific info for the part.
    Dim partDef As PartComponentDefinition
    Set partDef = partDoc.ComponentDefinition

    ' Iterate over the extrude features.
    Dim extrude As ExtrudeFeature
    For Each extrude In partDef.Features.ExtrudeFeatures
        Debug.Print extrude.name

        ' Get the definition object from the feature.
        Dim extrudeDef As ExtrudeDefinition
        Set extrudeDef = extrude.Definition

        ' Display information in the definition object.
        Select Case extrudeDef.ExtentType
            Case kDistanceExtent
                Dim distance As DistanceExtent
                Set distance = extrudeDef.Extent

                Debug.Print "Distance"
                Call DisplayToleranceInfo(distance.distance)
            Case kFromToExtent
                Dim fromTo As FromToExtent
                Set fromTo = extrudeDef.Extent

                Debug.Print "FromTo extent between to faces and/or work features."
            Case kThroughAllExtent
                Dim throughAll As ThroughAllExtent
                Set throughAll = extrudeDef.Extent

                Debug.Print "Through all extent."
            Case kToExtent
                Dim toExt As ToExtent
                Set toExt = extrudeDef.Extent

                Debug.Print "To a face or work plane extent."
            Case kToNextExtent
                Dim toNext As ToNextExtent
                Set toNext = extrudeDef.Extent

                Debug.Print "To next extent."
            Case Else
                Debug.Print "Unhandled case: " & extrudeDef.ExtentType
        End Select
    Next
End Sub

' Utility function used to display tolerance info for a parameter.
Private Sub DisplayToleranceInfo(Param As Parameter)
    ' Get the Tolerance object from the parameter.
    Dim tol As Tolerance
    Set tol = Param.Tolerance

    ' Display some info about the tolerance.
    Select Case tol.ToleranceType
        Case kDefaultTolerance
            Debug.Print "kDefaultTolerance"
        Case kBasicTolerance
            Debug.Print "kBasicTolerance"
        Case kDeviationTolerance
            Debug.Print "kDeviationTolerance"
        Case kLimitLinearTolerance
            Debug.Print "kLimitLinearTolerance"
        Case kLimitsFitsLinearTolerance
            Debug.Print "kLimitsFitsLinearTolerance"
        Case kLimitsFitsShowSizeTolerance
            Debug.Print "kLimitsFitsShowSizeTolerance"
        Case kLimitsFitsShowTolerance
            Debug.Print "kLimitsFitsShowTolerance"
        Case kLimitsFitsStackedTolerance
            Debug.Print "kLimitsFitsStackedTolerance"
        Case kLimitsStackedTolerance
            Debug.Print "kLimitsStackedTolerance"
        Case kMaxTolerance
            Debug.Print "kMaxTolerance"
        Case kMinTolerance
            Debug.Print "kMinTolerance"
        Case kOverrideTolerance
            Debug.Print "kOverrideTolerance"
        Case kReferenceTolerance
            Debug.Print "kReferenceTolerance"
        Case kSymmetricTolerance
            Debug.Print "kSymmetricTolerance"
    End Select

    Debug.Print "  Upper: " & tol.Upper
    Debug.Print "  Lower: " & tol.Lower
    Debug.Print "  Hole: " & tol.HoleTolerance
    Debug.Print "  Shaft: " & tol.ShaftTolerance
End Sub
```

Copy Code

```
Sub Main
    ' Get the active document assuming it is a part.
    Dim partDoc As PartDocument
    partDoc = ThisApplication.ActiveDocument

    ' Get the component definition.  This owns the part specific info for the part.
    Dim partDef As PartComponentDefinition
    partDef = partDoc.ComponentDefinition
```

```
    ' Iterate over the extrude features.
    Dim extrude As ExtrudeFeature
    For Each extrude In partDef.Features.ExtrudeFeatures
        Logger.Info(extrude.name)

        ' Get the definition object from the feature.
        Dim extrudeDef As ExtrudeDefinition
        extrudeDef = extrude.Definition

        ' Display information in the definition object.
        Select Case extrudeDef.ExtentType
            Case kDistanceExtent
                Dim distance As DistanceExtent
                distance = extrudeDef.Extent

                Logger.Info("Distance")

                Call DisplayToleranceInfo(distance.distance)
            Case kFromToExtent
                Dim fromTo As FromToExtent
                fromTo = extrudeDef.Extent

                Logger.Info("FromTo extent between to faces and/or work features.")

            Case kThroughAllExtent
                Dim throughAll As ThroughAllExtent
                throughAll = extrudeDef.Extent

                Logger.Info("Through all extent.")

            Case kToExtent
                Dim toExt As ToExtent
                toExt = extrudeDef.Extent

                Logger.Info("To a face or work plane extent.")

            Case kToNextExtent
                Dim toNext As ToNextExtent
                toNext = extrudeDef.Extent

                Logger.Info("To next extent.")

            Case Else
                Logger.Info("Unhandled case: " & extrudeDef.ExtentType)

        End Select
    Next
End Sub
' Utility function used to display tolerance info for a parameter.
Private Sub DisplayToleranceInfo(Param As Parameter)
    ' Get the Tolerance object from the parameter.
    Dim tol As Tolerance
    tol = Param.Tolerance

    ' Display some info about the tolerance.
    Select Case tol.ToleranceType
        Case kDefaultTolerance
            Logger.Info("kDefaultTolerance")
        Case kBasicTolerance
            Logger.Info("kBasicTolerance")
        Case kDeviationTolerance
            Logger.Info("kDeviationTolerance")
        Case kLimitLinearTolerance
            Logger.Info("kLimitLinearTolerance")
        Case kLimitsFitsLinearTolerance
            Logger.Info("kLimitsFitsLinearTolerance")
        Case kLimitsFitsShowSizeTolerance
            Logger.Info("kLimitsFitsShowSizeTolerance")
        Case kLimitsFitsShowTolerance
            Logger.Info("kLimitsFitsShowTolerance")
        Case kLimitsFitsStackedTolerance
            Logger.Info("kLimitsFitsStackedTolerance")
        Case kLimitsStackedTolerance
            Logger.Info("kLimitsStackedTolerance")
        Case kMaxTolerance
            Logger.Info("kMaxTolerance")
        Case kMinTolerance
            Logger.Info("kMinTolerance")
        Case kOverrideTolerance
            Logger.Info("kOverrideTolerance")
        Case kReferenceTolerance
            Logger.Info("kReferenceTolerance")
        Case kSymmetricTolerance
            Logger.Info("kSymmetricTolerance")
    End Select

    Logger.Info("  Upper: " & tol.Upper)
    Logger.Info("  Lower: " & tol.Lower)
    Logger.Info("  Hole: " & tol.HoleTolerance)
    Logger.Info("  Shaft: " & tol.ShaftTolerance)
End Sub
```

Extrude Feature - Create Block with Pocket

Description

This sample demonstrates creating a simple solid consisting a block with a pocket. It shows how to create a sketch plane at a specified orientation to existing geometry.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub DrawBlockWithPocket()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.  Since it's being created on
    ' one of the base workplanes we know the orientation it will be created in
    ' and don't need to worry about controlling it.  Because of this we also
    ' know the origin of the sketch plane will be at (0,0,0) in model space.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                                oTransGeom.CreatePoint2d(0, 0), _
                                oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    Set oFrontFace = oExtrude.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                  oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    Set oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    ' Create a profile.
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a pocket .25 cm deep.
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.  Since it's being created on
    ' one of the base workplanes we know the orientation it will be created in
    ' and don't need to worry about controlling it.  Because of this we also
    ' know the origin of the sketch plane will be at (0,0,0) in model space.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                                oTransGeom.CreatePoint2d(0, 0), _
                                oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
```

```
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    ' Create a profile.
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a pocket .25 cm deep.
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
```

Edit profile of an extrude feature

Description

This sample demonstrates editing the profile of an extrude feature.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub EditFeatureProfile()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    Dim oCenter As Point2d
    Set oCenter = oTransGeom.CreatePoint2d(-5, 0)

    ' Create a sketch circle
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius(oCenter, 1)

    ' Create a profile based on the circle
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 4 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(4, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Add all the lines of the rectangle to an ObjectCollection
    Dim oPathSegments As ObjectCollection
    Set oPathSegments = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oEntity As SketchEntity
    For Each oEntity In oRectangleLines
      oPathSegments.Add oEntity
    Next

    ' Create a profile that represents the newly created rectangle
    ' and excludes the original circle.
    Set oProfile = oSketch.Profiles.AddForSolid(False, oPathSegments)
```

```
    ' Set the new profile
    oExtrude.Definition.Profile = oProfile
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    Dim oCenter As Point2d
    oCenter = oTransGeom.CreatePoint2d(-5, 0)

    ' Create a sketch circle
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius(oCenter, 1)

    ' Create a profile based on the circle
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 4 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(4, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Add all the lines of the rectangle to an ObjectCollection
    Dim oPathSegments As ObjectCollection
    oPathSegments = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oEntity As SketchEntity
    For Each oEntity In oRectangleLines
      oPathSegments.Add(oEntity)
    Next

    ' Create a profile that represents the newly created rectangle
    ' and excludes the original circle.
    oProfile = oSketch.Profiles.AddForSolid(False, oPathSegments)

    ' Set the new profile
    oExtrude.Definition.Profile = oProfile
```

Extrude sketch text

Description

This sample demonstrates the creation of an extrude feature from sketch text.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ExtrudeSketchText()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a text box at (0,0)
    Dim oTextBox As TextBox
```

```
    Set oTextBox = oSketch.TextBoxes.AddFitted(oTransGeom.CreatePoint2d(0, 0), "Inventor")

    ' Add the text box to an object collection
    Dim oPaths As ObjectCollection
    Set oPaths = ThisApplication.TransientObjects.CreateObjectCollection
    oPaths.Add oTextBox

    ' Create a profile. Calling the AddForSolid method without any
    ' arguments will result in a profile containing all possible
    ' paths in the sketch. By passing in the text box, the profile
    ' is restricted to the input text path.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid(False, oPaths)

    ' Extrude the text.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a text box at (0,0)
    Dim oTextBox As TextBox
    oTextBox = oSketch.TextBoxes.AddFitted(oTransGeom.CreatePoint2d(0, 0), "Inventor")

    ' Add the text box to an object collection
    Dim oPaths As ObjectCollection
    oPaths = ThisApplication.TransientObjects.CreateObjectCollection
    oPaths.Add(oTextBox)

    ' Create a profile. Calling the AddForSolid method without any
    ' arguments will result in a profile containing all possible
    ' paths in the sketch. By passing in the text box, the profile
    ' is restricted to the input text path.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid(False, oPaths)

    ' Extrude the text.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
```

Fillet Feature (All Rounds)

Description

This sample demonstrates rounding all of the edges of a part.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateAllRoundsFillet()
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                          ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                          ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
```

```
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Declare an EdgeCollection object to pass in since it is a required argument,
    ' but since we're going to specify all rounds, we don't need to actually create
    ' the EdgeCollection or add any edges to it.
    Dim oEdges As EdgeCollection

    ' Create the fillet feature specifying to do all rounds.
    Dim oFillet As FilletFeature
    Set oFillet = oCompDef.Features.FilletFeatures.AddSimple(oEdges, 1, , True)
End Sub
```

Copy Code

```
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                          ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                          ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Declare an EdgeCollection object to pass in since it is a required argument,
    ' but since we're going to specify all rounds, we don't need to actually create
    ' the EdgeCollection or add any edges to it.
    Dim oEdges As EdgeCollection

    ' Create the fillet feature specifying to do all rounds.
    Dim oFillet As FilletFeature
    oFillet = oCompDef.Features.FilletFeatures.AddSimple(oEdges, 1, , True)
```

Fillet Feature (Complex)

Description

This sample demonstrates creating a complex fillet. The result in this case has several different constant radii fillets and two edges that use variable radius, with one of these having a different radius defined along the edge.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateFilletComplex()
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Dim oEnts As SketchEntitiesEnumerator
    Set oEnts = oSketch.SketchLines.AddAsTwoPointRectangle( _
                          ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                          ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
```

```
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(8, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create an EdgeCollection object to use for inputting the edges for fillets.
    Dim oEdgeCollection As EdgeCollection
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Add all of the edges of the start face to the edge collection.
    Dim oEdge As Edge
    For Each oEdge In oExtrude.StartFaces.Item(1).Edges
        oEdgeCollection.Add oEdge
    Next

    ' Obtain a FilletDefinition object to use in defining the various inputs to create the fillet.
    Dim oFilletDef As FilletDefinition
    Set oFilletDef = oCompDef.Features.FilletFeatures.CreateFilletDefinition

    ' Create the first edge set.
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 1.5)

    ' Reinitialize the edge collection.
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Add all of the edges of the end face to the edge collection.
    For Each oEdge In oExtrude.EndFaces.Item(1).Edges
        oEdgeCollection.Add oEdge
    Next

    ' Create the first edge set.
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 1)

    ' Find the edges that go between the start and end faces by checking to
    ' see if they're parallel to the Z axis.
    Dim oZVector As UnitVector
    Set oZVector = ThisApplication.TransientGeometry.CreateUnitVector(0, 0, 1)
    Dim oSideEdges(1 To 4) As Edge
    Dim EdgeCount As Long
    EdgeCount = 0
    For Each oEdge In oCompDef.SurfaceBodies.Item(1).Edges
        ' In this case we know all the edges are linear.
        Dim oLine As LineSegment
        Set oLine = oEdge.Geometry

        If oLine.Direction.IsParallelTo(oZVector) Then
            EdgeCount = EdgeCount + 1
            Set oSideEdges(EdgeCount) = oEdge
        End If
    Next

    ' Add the first two edges to a constant radius edge set.
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add oSideEdges(1)
    oEdgeCollection.Add oSideEdges(2)
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 0.5)

    ' Create a variable radius edge set for the third edge.
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add oSideEdges(3)
    Call oFilletDef.AddVariableRadiusEdgeSet(oEdgeCollection, 1, 2)

    ' Create a variable radius edge with a different internal radius for the fourth edge.
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add oSideEdges(4)
    Dim oVarRadiusEdgeSet As FilletVariableRadiusEdgeSet
    Set oVarRadiusEdgeSet = oFilletDef.AddVariableRadiusEdgeSet(oEdgeCollection, 0.5, 1.5)
    Call oVarRadiusEdgeSet.AddIntermediateRadius(oSideEdges(4), 3, 0.5)

    Dim oFillet As FilletFeature
    Set oFillet = oCompDef.Features.FilletFeatures.Add(oFilletDef, False)
End Sub
```

Copy Code

```
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Dim oEnts As SketchEntitiesEnumerator
    oEnts = oSketch.SketchLines.AddAsTwoPointRectangle( _
                            ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                            ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(8, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create an EdgeCollection object to use for inputting the edges for fillets.
    Dim oEdgeCollection As EdgeCollection
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
```

```
    ' Add all of the edges of the start face to the edge collection.
    Dim oEdge As Edge
    For Each oEdge In oExtrude.StartFaces.Item(1).Edges
        oEdgeCollection.Add(oEdge)
    Next

    ' Obtain a FilletDefinition object to use in defining the various inputs to create the fillet.
    Dim oFilletDef As FilletDefinition
    oFilletDef = oCompDef.Features.FilletFeatures.CreateFilletDefinition

    ' Create the first edge set.
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 1.5)

    ' Reinitialize the edge collection.
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Add all of the edges of the end face to the edge collection.
    For Each oEdge In oExtrude.EndFaces.Item(1).Edges
        oEdgeCollection.Add(oEdge)
    Next

    ' Create the first edge set.
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 1)

    ' Find the edges that go between the start and end faces by checking to
    ' see if they're parallel to the Z axis.
    Dim oZVector As UnitVector
    oZVector = ThisApplication.TransientGeometry.CreateUnitVector(0, 0, 1)
    Dim oSideEdges(0 To 3) As Edge
    Dim EdgeCount As Long
    EdgeCount = 0
    For Each oEdge In oCompDef.SurfaceBodies.Item(1).Edges
        ' In this case we know all the edges are linear.
        Dim oLine As LineSegment
        oLine = oEdge.Geometry

        If oLine.Direction.IsParallelTo(oZVector) Then
            EdgeCount = EdgeCount + 1
            oSideEdges(EdgeCount - 1) = oEdge
        End If
    Next

    ' Add the first two edges to a constant radius edge set.
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add(oSideEdges(0))
    oEdgeCollection.Add(oSideEdges(1))
    Call oFilletDef.AddConstantRadiusEdgeSet(oEdgeCollection, 0.5)

    ' Create a variable radius edge set for the third edge.
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add(oSideEdges(2))
    Call oFilletDef.AddVariableRadiusEdgeSet(oEdgeCollection, 1, 2)

    ' Create a variable radius edge with a different internal radius for the fourth edge.
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection
    oEdgeCollection.Add(oSideEdges(3))
    Dim oVarRadiusEdgeSet As FilletVariableRadiusEdgeSet
    oVarRadiusEdgeSet = oFilletDef.AddVariableRadiusEdgeSet(oEdgeCollection, 0.5, 1.5)
    Call oVarRadiusEdgeSet.AddIntermediateRadius(oSideEdges(3), 3, 0.5)

    Dim oFillet As FilletFeature
    oFillet = oCompDef.Features.FilletFeatures.Add(oFilletDef, False)
```

Fillet Feature (Simple)

Description

This sample demonstrates using the AddSimple method of the FilletFeatures collection to create a constant radius fillet.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSimpleFillet()
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                          ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                          ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid
```

```
    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Define the set of edges that are for the start and end faces of the solid.
    Dim oEdges As EdgeCollection
    Set oEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oEdge As Edge
    For Each oEdge In oExtrude.StartFaces.Item(1).Edges
        oEdges.Add oEdge
    Next

    For Each oEdge In oExtrude.EndFaces.Item(1).Edges
        oEdges.Add oEdge
    Next

    ' Create the fillet feature.
    Dim oFillet As FilletFeature
    Set oFillet = oCompDef.Features.FilletFeatures.AddSimple(oEdges, 1)
End Sub
```

Copy Code

```
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a sketch on the xy work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a rectangle.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                            ThisApplication.TransientGeometry.CreatePoint2d(-6, -4), _
                            ThisApplication.TransientGeometry.CreatePoint2d(6, 4))

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kSymmetricExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Define the set of edges that are for the start and end faces of the solid.
    Dim oEdges As EdgeCollection
    oEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oEdge As Edge
    For Each oEdge In oExtrude.StartFaces.Item(1).Edges
        oEdges.Add(oEdge)
    Next

    For Each oEdge In oExtrude.EndFaces.Item(1).Edges
        oEdges.Add(oEdge)
    Next

    ' Create the fillet feature.
    Dim oFillet As FilletFeature
    oFillet = oCompDef.Features.FilletFeatures.AddSimple(oEdges, 1)
```

Finish Feature Creation

Description

This sample demonstrates how to create a finish feature.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a finish feature.

Copy Code

```
Sub FinishFeatureSample()
    ' Create a new part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
```

```
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    Set oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    Set oFrontFace = oExtrude1.StartFaces.Item(1)


    Dim oGeometries As FaceCollection
    Set oGeometries = ThisApplication.TransientObjects.CreateFaceCollection

    oGeometries.Add oFrontFace


    Dim oFinishFeatures As FinishFeatures
    Set oFinishFeatures = oCompDef.Features.FinishFeatures

    Dim oAppearance As Asset
    ' Get an appearance asset from library: Autodesk Appearance Library
    ThisApplication.AssetLibraries("314DE259-5443-4621-BFBD-1730C6CC9AE9").AppearanceAssets("Red").CopyTo oPartDoc
    Set oAppearance = oPartDoc.AppearanceAssets.Item("Red")

    ' Create finish definition.
    Dim oFinishDef As FinishDefinition
    Set oFinishDef = oFinishFeatures.CreateFinishDefinition(oGeometries, FinishTypeEnum.kAppearanceFinishType, , oAppearance)

    ' Create finish feature.
    Dim oFinish As FinishFeature
    Set oFinish = oFinishFeatures.Add(oFinishDef)

End Sub
```

This sample demonstrates how to create a finish feature.

Copy Code

```
    ' Create a new part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude1.StartFaces.Item(1)


    Dim oGeometries As FaceCollection
    oGeometries = ThisApplication.TransientObjects.CreateFaceCollection

    oGeometries.Add(oFrontFace)


    Dim oFinishFeatures As FinishFeatures
    oFinishFeatures = oCompDef.Features.FinishFeatures

    Dim oAppearance As Asset
    ' Get an appearance asset from library: Autodesk Appearance Library
    ThisApplication.AssetLibraries("314DE259-5443-4621-BFBD-1730C6CC9AE9").AppearanceAssets("Red").CopyTo(oPartDoc)
    oAppearance = oPartDoc.AppearanceAssets.Item("Red")
```

```
    ' Create finish definition.
    Dim oFinishDef As FinishDefinition
    oFinishDef = oFinishFeatures.CreateFinishDefinition(oGeometries, FinishTypeEnum.kAppearanceFinishType, , oAppearance)

    ' Create finish feature.
    Dim oFinish As FinishFeature
    oFinish = oFinishFeatures.Add(oFinishDef)
```

Highlight Feature Faces

Description

This sample highlights the faces of an extrusion, revolution, or hole feature. It differentiates the faces on the start cap, end cap, and side faces by highlighting them in different colors. The HighlightFeatureFaces sub highlights the feature faces. Since the highlight set objects are declared outside of this sub, the highlighting remains after the sub has finished executing. Use the ClearHighlight sub to clear the highlighting that does so by releasing the HighlightSet objects.

Code Samples

- VBA
- iLogic

Copy Code

```
Private oStartHLSet As HighlightSet
Private oEndHLSet As HighlightSet
Private oSideHLSet As HighlightSet

Public Sub HighlightFeatureFaces()
    ' Get the first item in the select set. This assumes it is a feature.
    If ThisApplication.ActiveDocument.SelectSet.Count > 0 Then
        Dim oFeature As PartFeature
        Set oFeature = ThisApplication.ActiveDocument.SelectSet.Item(1)
    Else
        MsgBox "You must select a feature."
        Exit Sub
    End If

    ' Check to see that it's an extrusion, revolution or hole feature.
    If oFeature.Type <> kExtrudeFeatureObject And _
       oFeature.Type <> kRevolveFeatureObject And _
       oFeature.Type <> kHoleFeatureObject Then
        MsgBox "You must select an extrusion, revolution, or hole."
        Exit Sub
    End If

    ' Create a new highlight set for the start face(s).
    Set oStartHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

    ' Change the highlight color for the set to red.
    Dim oRed As Color
    Set oRed = ThisApplication.TransientObjects.CreateColor(255, 0, 0)

    ' Set the opacity
    oRed.Opacity = 0.8

    oStartHLSet.Color = oRed

    ' Add all start faces to the highlightset. Skip holes because
    ' they don't support the StartFaces property.
    If oFeature.Type <> kHoleFeatureObject Then
        Dim oFace As Face
        For Each oFace In oFeature.StartFaces
            oStartHLSet.AddItem oFace
        Next
    End If

    ' Create a new highlight set for the end face(s).
    Set oEndHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

    ' Change the highlight color for the set to green.
    Dim oGreen As Color
    Set oGreen = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

    oEndHLSet.Color = oGreen

    ' Add all end faces to the highlightset.
    For Each oFace In oFeature.EndFaces
        oEndHLSet.AddItem oFace
    Next

    ' Create a new highlight set for the side face(s).
    Set oSideHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

    ' Change the highlight color for the set to blue.
    Dim oBlue As Color
    Set oBlue = ThisApplication.TransientObjects.CreateColor(0, 0, 255)

    oSideHLSet.Color = oBlue

    ' Add all end faces to the highlightset.
    For Each oFace In oFeature.SideFaces
        oSideHLSet.AddItem oFace
    Next

    MsgBox "Start faces are displayed in red." & Chr(13) & _
           "End faces are displayed in green." & Chr(13) & _
```

```
            "Side faces are displayed in blue.", vbOKOnly + vbInformation
End Sub

Public Sub ClearHighlight()

    ' Release the highlight set objects to clear highlighting.
    ' Alternatively, HighlightSet.Delete or HighlightSet.Clear
    ' can also be used to clear the highlighting.

    Set oStartHLSet = Nothing
    Set oEndHLSet = Nothing
    Set oSideHLSet = Nothing
End Sub
```

Copy Code

```
Class Test
        Private oStartHLSet As HighlightSet
        Private oEndHLSet As HighlightSet
        Private oSideHLSet As HighlightSet
        Private oFeature As PartFeature
        Sub Main
            ' Get the first item in the select set. This assumes it is a feature.
            If ThisApplication.ActiveDocument.SelectSet.Count > 0 Then
                oFeature = ThisApplication.ActiveDocument.SelectSet.Item(1)
            Else
                MsgBox("You must select a feature.")
                Exit Sub
            End If

            ' Check to see that it's an extrusion, revolution or hole feature.
            If oFeature.Type <> kExtrudeFeatureObject And _
               oFeature.Type <> kRevolveFeatureObject And _
               oFeature.Type <> kHoleFeatureObject Then
                MsgBox("You must select an extrusion, revolution, or hole.")
                Exit Sub
            End If

            ' Create a new highlight set for the start face(s).
            oStartHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

            ' Change the highlight color for the set to red.
            Dim oRed As Color
            oRed = ThisApplication.TransientObjects.CreateColor(255, 0, 0)

            ' Set the opacity
            oRed.Opacity = 0.8

            oStartHLSet.Color = oRed

            ' Add all start faces to the highlightset. Skip holes because
            ' they don't support the StartFaces property.
            If oFeature.Type <>  kHoleFeatureObject Then
                Dim oFace As Face
                For Each oFace In oFeature.StartFaces
                    oStartHLSet.AddItem(oFace)
                Next
            End If

            ' Create a new highlight set for the end face(s).
            oEndHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

            ' Change the highlight color for the set to green.
            Dim oGreen As Color
            oGreen = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

            oEndHLSet.Color = oGreen

            ' Add all end faces to the highlightset.
            For Each oFace In oFeature.EndFaces
                oEndHLSet.AddItem(oFace)
            Next

            ' Create a new highlight set for the side face(s).
            oSideHLSet = ThisApplication.ActiveDocument.CreateHighlightSet

            ' Change the highlight color for the set to blue.
            Dim oBlue As Color
            oBlue = ThisApplication.TransientObjects.CreateColor(0, 0, 255)

            oSideHLSet.Color = oBlue

            ' Add all end faces to the highlightset.
            For Each oFace In oFeature.SideFaces
                oSideHLSet.AddItem(oFace)
            Next

            MsgBox("Start faces are displayed in red." & Chr(13) & _
                   "End faces are displayed in green." & Chr(13) & _
                   "Side faces are displayed in blue.", vbOKOnly + vbInformation)
        End Sub

        Public Sub ClearHighlight()

            ' Release the highlight set objects to clear highlighting.
            ' Alternatively, HighlightSet.Delete or HighlightSet.Clear
            ' can also be used to clear the highlighting.

            oStartHLSet = Nothing
            oEndHLSet = Nothing
            oSideHLSet = Nothing
        End Sub
End Class
```

Hole Feature - Through holes (RegularAndTapped)

Description

This sample demonstrates the creation of through holes, both regular and tapped.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub HoleSample()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a rectangle on the sketch.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                                    oTransGeom.CreatePoint2d(0, 0), _
                                    oTransGeom.CreatePoint2d(6, 4))

    ' Create the profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent("2 cm", kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a new sketch to contain the points that define the hole centers.
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create an object collection for the hole center points.
    Dim oHoleCenters As ObjectCollection
    Set oHoleCenters = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add two points as hole centers.
    oHoleCenters.Add oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(1, 1))
    oHoleCenters.Add oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(5, 1))

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                            oHoleCenters, "1 cm", kPositiveExtentDirection)

    ' Define tap information.

    Dim oHoleTapInfo As HoleTapInfo
    Set oHoleTapInfo = oCompDef.Features.HoleFeatures.CreateTapInfo( _
                            True, "ANSI Unified Screw Threads", _
                            "7/16-14 UNC", "1B", False, "1 cm")

    ' Create a new sketch for the tapped hole centers.
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create a new object collection for the hole center points.
    Set oHoleCenters = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add two points as hole centers.
    oHoleCenters.Add oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(1, 3))
    oHoleCenters.Add oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(5, 3))

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                            oHoleCenters, oHoleTapInfo, kPositiveExtentDirection)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
```

```
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a rectangle on the sketch.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                                    oTransGeom.CreatePoint2d(0, 0), _
                                    oTransGeom.CreatePoint2d(6, 4))

    ' Create the profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent("2 cm", kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a new sketch to contain the points that define the hole centers.
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create an object collection for the hole center points.
    Dim oHoleCenters As ObjectCollection
    oHoleCenters = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add two points as hole centers.
    oHoleCenters.Add(oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(1, 1)))
    oHoleCenters.Add(oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(5, 1)))

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                           oHoleCenters, "1 cm", kPositiveExtentDirection)

    ' Define tap information.

    Dim oHoleTapInfo As HoleTapInfo
    oHoleTapInfo = oCompDef.Features.HoleFeatures.CreateTapInfo( _
                           True, "ANSI Unified Screw Threads", _
                           "7/16-14 UNC", "1B", False, "1 cm")

    ' Create a new sketch for the tapped hole centers.
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create a new object collection for the hole center points.
    oHoleCenters = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add two points as hole centers.
    oHoleCenters.Add(oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(1, 3)))
    oHoleCenters.Add(oSketch.SketchPoints.Add(oTransGeom.CreatePoint2d(5, 3)))

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                           oHoleCenters, oHoleTapInfo, kPositiveExtentDirection)
```

Hole feature linear placement

Description

This sample demonstrates the creation of a hole feature using the linear placement type.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub HoleFeatureLinearPlacement()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a square on the sketch.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                                    oTransGeom.CreatePoint2d(0, 0), _
                                    oTransGeom.CreatePoint2d(6, 6))

    ' Create the profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent("2 cm", kNegativeExtentDirection)
```

```
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the start face of the extrude.
    Dim oFace As Face
    Set oFace = oExtrude.StartFaces(1)

    ' Get two adjacent edges on the start face.
    Dim oEdge1, oEdge2 As Edge
    Set oEdge1 = oFace.Edges(1)
    Set oEdge2 = oFace.Edges(2)

    ' Create a bias point for hole placement to place it at
    ' the expected location. This is the model point
    ' corresponding to the center of the square in the sketch.
    Dim oBiasPoint As Point
    Set oBiasPoint = oSketch.SketchToModelSpace(oTransGeom.CreatePoint2d(1.5, 1.5))

    ' Create the hole feature placement definition.
    Dim oLinearPlacementDef As LinearHolePlacementDefinition
    Set oLinearPlacementDef = oCompDef.Features.HoleFeatures.CreateLinearPlacementDefinition _
    (oFace, oEdge1, "2 cm", oEdge2, "2 cm", oBiasPoint)

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                          oLinearPlacementDef, "1 cm", kPositiveExtentDirection)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create a square on the sketch.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                                      oTransGeom.CreatePoint2d(0, 0), _
                                      oTransGeom.CreatePoint2d(6, 6))

    ' Create the profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrusion.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent("2 cm", kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the start face of the extrude.
    Dim oFace As Face
    oFace = oExtrude.StartFaces(1)

    ' Get two adjacent edges on the start face.
    Dim oEdge1, oEdge2 As Edge
    oEdge1 = oFace.Edges(1)
    oEdge2 = oFace.Edges(2)

    ' Create a bias point for hole placement to place it at
    ' the expected location. This is the model point
    ' corresponding to the center of the square in the sketch.
    Dim oBiasPoint As Point
    oBiasPoint = oSketch.SketchToModelSpace(oTransGeom.CreatePoint2d(1.5, 1.5))

    ' Create the hole feature placement definition.
    Dim oLinearPlacementDef As LinearHolePlacementDefinition
    oLinearPlacementDef = oCompDef.Features.HoleFeatures.CreateLinearPlacementDefinition _
    (oFace, oEdge1, "2 cm", oEdge2, "2 cm", oBiasPoint)

    ' Create the hole feature.
    Call oCompDef.Features.HoleFeatures.AddDrilledByThroughAllExtent( _
                          oLinearPlacementDef, "1 cm", kPositiveExtentDirection)
```

Placement of a standard iFeature

Description

This program demonstrates the placement of a standard iFeature in a part.

Code Samples

- VBA
- iLogic

A part must be open and a planar face within that part selected. The iFeature used is delivered as a sample with Inventor.

Copy Code

```
Public Sub PlaceiFeature()
    ' Get the part document and component definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    If Err Then
        MsgBox "A part must be active."
        Exit Sub
    End If

    Dim oPartDef As PartComponentDefinition
    Set oPartDef = oPartDoc.ComponentDefinition

    ' Get the selected face to use as input for the iFeature.
    Dim oFace As Face
    Set oFace = oPartDoc.SelectSet.Item(1)
    If Err Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType  kPlaneSurface Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    Set oFeatures = oPartDef.Features

    ' Create an iFeatureDefinition object, update the file path if necessary.
    Dim oiFeatureDef As iFeatureDefinition
    Set oiFeatureDef = oFeatures.iFeatures.CreateiFeatureDefinition( _
"C:\Users\Public\Documents\Autodesk\Inventor 2025\Catalog\Slots\End_mill_curved.ide")

    ' Set the input.
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Sketch Plane"
                Dim oPlaneInput As iFeatureSketchPlaneInput
                Set oPlaneInput = oInput
                oPlaneInput.PlaneInput = oFace
            Case "Diameter"
                Set oParamInput = oInput
                oParamInput.Expression = "1 in"
            Case "Depth"
                Set oParamInput = oInput
                oParamInput.Expression = "0.5 in"
        End Select
    Next

    ' Create the iFeature.
    Dim oiFeature As iFeature
    Set oiFeature = oFeatures.iFeatures.Add(oiFeatureDef)
End Sub
```

A part must be open and a planar face within that part selected. The iFeature used is delivered as a sample with Inventor.

Copy Code

```
    ' Get the part document and component definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    If Err.Number > 0 Then
        MsgBox("A part must be active.")
        Exit Sub
    End If

    Dim oPartDef As PartComponentDefinition
    oPartDef = oPartDoc.ComponentDefinition

    ' Get the selected face to use as input for the iFeature.
    Dim oFace As Face
    oFace = oPartDoc.SelectSet.Item(1)
    If Err.Number > 0  Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType <> kPlaneSurface Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    oFeatures = oPartDef.Features

    ' Create an iFeatureDefinition object, update the file path if necessary.
    Dim oiFeatureDef As iFeatureDefinition
    oiFeatureDef = oFeatures.iFeatures.CreateiFeatureDefinition( _
"C:\Users\Public\Documents\Autodesk\Inventor 2025\Catalog\Slots\End_mill_curved.ide")

    ' Set the input.
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Sketch Plane"
                Dim oPlaneInput As iFeatureSketchPlaneInput
                oPlaneInput = oInput
```

```
            oPlaneInput.PlaneInput = oFace
        Case "Diameter"
            oParamInput = oInput
            oParamInput.Expression = "1 in"
        Case "Depth"
            oParamInput = oInput
            oParamInput.Expression = "0.5 in"
    End Select
Next

' Create the iFeature.
Dim oiFeature As iFeature
oiFeature = oFeatures.iFeatures.Add(oiFeatureDef)
```

Place table driven iFeature

Description

This program demonstrates the placement of a table driven iFeature in a part.

Code Samples

- VBA
- iLogic

A part must be open and a planar face on that part selected. There aren't any table driven .ide files delivered with Inventor so one must be constructed to be used as input. The primary assumption this sample makes about the .ide file is that a custom row has been created named 'Size' and one of the rows in this column has the value '4.5.'

Copy Code

```
Public Sub PlaceTableDriveniFeature()
    ' Get the part document and component definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    If Err Then
        MsgBox "A part must be active."
        Exit Sub
    End If

    Dim oPartDef As PartComponentDefinition
    Set oPartDef = oPartDoc.ComponentDefinition

    ' Get the selected face to use as input for the iFeature.
    Dim oFace As Face
    Set oFace = oPartDoc.SelectSet.Item(1)
    If Err Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType <> kPlaneSurface Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    Set oFeatures = oPartDef.Features

    ' Create an iFeatureDefinition object.
    Dim oiFeatureDef As iFeatureDefinition
    Set oiFeatureDef = oFeatures.iFeatures.CreateiFeatureDefinition( _
"C:\Users\Public\Documents\Autodesk\Inventor 2026\Catalog\Table\Test.ide")

    ' Set the input, which in this case is only the sketch plane
    ' since the other input comes from the table.  The parameter input
    ' should not be available at this point since it can't be changed
    ' and is controlled by the table.
    '
    ' When an existing table driven iFeature is accessed then this should
    ' include the parameters so the programmer has access to the corresponding
    ' reference parameter that's created.
    Dim bFoundPlaneInput As Boolean
    bFoundPlaneInput = False
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Profile Plane1"
                Dim oPlaneInput As iFeatureSketchPlaneInput
                Set oPlaneInput = oInput
                oPlaneInput.PlaneInput = oFace
                bFoundPlaneInput = True
        End Select
    Next

    If Not bFoundPlaneInput Then
        MsgBox "The ide file does not contain an iFeature input named ""Profile Plane1""."
        Exit Sub
    End If

    '** Look through the table to find the row where "Size" is "4.5".
    Dim oTable As iFeatureTable
    Set oTable = oiFeatureDef.iFeatureTable

    ' Find the "Size" column.
```

```
    Dim oColumn As iFeatureTableColumn
    Dim bFoundSize As Boolean
    bFoundSize = False
    For Each oColumn In oTable.iFeatureTableColumns
        If oColumn.DisplayHeading = "Size" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox "The column ""Size"" was not found."
        Exit Sub
    End If

    ' Find the row in the "Size" column with the value of "4.5"
    Dim oCell As iFeatureTableCell
    bFoundSize = False
    For Each oCell In oColumn
        If oCell.Value = "4.5" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox "The cell with value ""4.5"" was not found."
        Exit Sub
    End If

    ' Set this row as the active row.
    oiFeatureDef.ActiveTableRow = oCell.Row

    ' Create the iFeature.
    Dim oiFeature As iFeature
    Set oiFeature = oFeatures.iFeatures.Add(oiFeatureDef)
End Sub
```

A part must be open and a planar face on that part selected. There aren't any table driven .ide files delivered with Inventor so one must be constructed to be used as input. The primary assumption this sample makes about the .ide file is that a custom row has been created named 'Size' and one of the rows in this column has the value '4.5.'

Copy Code

```
    ' Get the part document and component definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    If Err.Number > 0 Then
        MsgBox("A part must be active.")
        Exit Sub
    End If

    Dim oPartDef As PartComponentDefinition
    oPartDef = oPartDoc.ComponentDefinition

    ' Get the selected face to use as input for the iFeature.
    Dim oFace As Face
    oFace = oPartDoc.SelectSet.Item(1)
    If Err.Number > 0 Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType <> kPlaneSurface Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    oFeatures = oPartDef.Features

    ' Create an iFeatureDefinition object.
    Dim oiFeatureDef As iFeatureDefinition
    oiFeatureDef = oFeatures.iFeatures.CreateiFeatureDefinition( _
"C:\Users\Public\Documents\Autodesk\Inventor 2026\Catalog\Table\Test.ide")

    ' Set the input, which in this case is only the sketch plane
    ' since the other input comes from the table.  The parameter input
    ' should not be available at this point since it can't be changed
    ' and is controlled by the table.
    '
    ' When an existing table driven iFeature is accessed then this should
    ' include the parameters so the programmer has access to the corresponding
    ' reference parameter that's created.
    Dim bFoundPlaneInput As Boolean
    bFoundPlaneInput = False
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Profile Plane1"
                Dim oPlaneInput As iFeatureSketchPlaneInput
                oPlaneInput = oInput
                oPlaneInput.PlaneInput = oFace
                bFoundPlaneInput = True
        End Select
    Next

    If Not bFoundPlaneInput Then
        MsgBox("The ide file does not contain an iFeature input named ""Profile Plane1""")
        Exit Sub
    End If

    '** Look through the table to find the row where "Size" is "4.5".
    Dim oTable As iFeatureTable
```

```
    oTable = oiFeatureDef.iFeatureTable

    ' Find the "Size" column.
    Dim oColumn As iFeatureTableColumn
    Dim bFoundSize As Boolean
    bFoundSize = False
    For Each oColumn In oTable.iFeatureTableColumns
        If oColumn.DisplayHeading = "Size" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox("The column ""Size"" was not found.")
        Exit Sub
    End If

    ' Find the row in the "Size" column with the value of "4.5"
    Dim oCell As iFeatureTableCell
    bFoundSize = False
    For Each oCell In oColumn
        If oCell.Value = "4.5" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox("The cell with value ""4.5"" was not found.")
        Exit Sub
    End If

    ' Set this row as the active row.
    oiFeatureDef.ActiveTableRow = oCell.Row

    ' Create the iFeature.
    Dim oiFeature As iFeature
    oiFeature = oFeatures.iFeatures.Add(oiFeatureDef)
```

Changing row of table driven iFeature

Description

This program demonstrates the edit of a table driven iFeature to change which row of the table is being used to drive the iFeature.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub EditTableDriveniFeature()
    ' Get the part document and component definition of the active document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    If Err Then
        MsgBox "A part must be active."
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    Set oFeatures = oPartDoc.ComponentDefinition.Features

    ' Get the first iFeature assuming that it is table-driven.
    Dim oiFeature As iFeature
    Set oiFeature = oFeatures.iFeatures.Item(1)

    ' Check to see if the first iFeature is table-driven.
    If Not oiFeature.iFeatureDefinition.IsTableDriven Then
        MsgBox "The first iFeature in the part is not table-driven."
        Exit Sub
    End If

    '** Look through the table to find the row where "Size" is "3".
    Dim oTable As iFeatureTable
    Set oTable = oiFeature.iFeatureDefinition.iFeatureTable

    ' Find the "Size" column.
    Dim oColumn As iFeatureTableColumn
    Dim bFoundSize As Boolean
    bFoundSize = False
    For Each oColumn In oTable.iFeatureTableColumns
        If oColumn.DisplayHeading = "Size" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox "The column ""Size"" was not found."
        Exit Sub
    End If

    ' Find the row in the "Size" column with the value of "3"
    Dim oCell As iFeatureTableCell
    bFoundSize = False
```

```
    For Each oCell In oColumn
        If oCell.Value = "3" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox "The cell with value ""3"" was not found."
        Exit Sub
    End If

    oiFeature.iFeatureDefinition.ActiveTableRow = oCell.Row
End Sub
```

Copy Code

```
    ' Get the part document and component definition of the active document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    If Err.Number > 0 Then
        MsgBox("A part must be active.")
        Exit Sub
    End If

    Dim oFeatures As PartFeatures
    oFeatures = oPartDoc.ComponentDefinition.Features

    ' Get the first iFeature assuming that it is table-driven.
    Dim oiFeature As iFeature
    oiFeature = oFeatures.iFeatures.Item(1)

    ' Check to see if the first iFeature is table-driven.
    If Not oiFeature.iFeatureDefinition.IsTableDriven Then
        MsgBox("The first iFeature in the part is not table-driven.")
        Exit Sub
    End If

    '** Look through the table to find the row where "Size" is "3".
    Dim oTable As iFeatureTable
    oTable = oiFeature.iFeatureDefinition.iFeatureTable

    ' Find the "Size" column.
    Dim oColumn As iFeatureTableColumn
    Dim bFoundSize As Boolean
    bFoundSize = False
    For Each oColumn In oTable.iFeatureTableColumns
        If oColumn.DisplayHeading = "Size" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox("The column ""Size"" was not found.")
        Exit Sub
    End If

    ' Find the row in the "Size" column with the value of "3"
    Dim oCell As iFeatureTableCell
    bFoundSize = False
    For Each oCell In oColumn
        If oCell.Value = "3" Then
            bFoundSize = True
            Exit For
        End If
    Next

    If Not bFoundSize Then
        MsgBox("The cell with value ""3"" was not found.")
        Exit Sub
    End If

    oiFeature.iFeatureDefinition.ActiveTableRow = oCell.Row
```

Adding a new stitch (knit) feature

Description

This sample demonstrates the creation of a stitch feature (known as the Knit feature in the API). The sample creates two work surfaces using surface extrusions and stitches them together.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub StitchFeatureCreate()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
```

```
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane. Dim oSketch As PlanarSketch
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw two sketch lines that will be used as the
    ' open profiles for creating work surfaces.
    Dim oLineOne As SketchLine
    Set oLineOne = oSketch.SketchLines.AddByTwoPoints( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(2, -2))

    Dim oLineTwo As SketchLine
    Set oLineTwo = oSketch.SketchLines.AddByTwoPoints( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(-2, -2))

    ' Create a profile for the first extrude.
    Dim oProfileOne As Profile
    Set oProfileOne = oSketch.Profiles.AddForSurface(oLineOne)

    ' Create an surface extrusion 2 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfileOne, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(2, kNegativeExtentDirection)
    Dim oExtrudeOne As ExtrudeFeature
    Set oExtrudeOne = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a profile for the second extrude.
    Dim oProfileTwo As Profile
    Set oProfileTwo = oSketch.Profiles.AddForSurface(oLineTwo)

    ' Create an extrusion 2 cm thick.
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfileTwo, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(2, kNegativeExtentDirection)
    Dim oExtrudeTwo As ExtrudeFeature
    Set oExtrudeTwo = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oSurfaces As ObjectCollection
    Set oSurfaces = ThisApplication.TransientObjects.CreateObjectCollection

    ' Since we know that the only work surfaces in the document
    ' are those created by the above extrusions, use those.
    oSurfaces.Add oCompDef.WorkSurfaces.Item(1)
    oSurfaces.Add oCompDef.WorkSurfaces.Item(2)

    ' Create a stitch (knit) feature by stitching
    ' together the two work surfaces created above.
    Dim oKnitFeature As KnitFeature
    Set oKnitFeature = oCompDef.Features.KnitFeatures.Add(oSurfaces)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
               ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane. Dim oSketch As PlanarSketch
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw two sketch lines that will be used as the
    ' open profiles for creating work surfaces.
    Dim oLineOne As SketchLine
    oLineOne = oSketch.SketchLines.AddByTwoPoints( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(2, -2))

    Dim oLineTwo As SketchLine
    oLineTwo = oSketch.SketchLines.AddByTwoPoints( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(-2, -2))

    ' Create a profile for the first extrude.
    Dim oProfileOne As Profile
    oProfileOne = oSketch.Profiles.AddForSurface(oLineOne)

    ' Create an surface extrusion 2 cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfileOne, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(2, kNegativeExtentDirection)
    Dim oExtrudeOne As ExtrudeFeature
    oExtrudeOne = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create a profile for the second extrude.
    Dim oProfileTwo As Profile
    oProfileTwo = oSketch.Profiles.AddForSurface(oLineTwo)

    ' Create an extrusion 2 cm thick.
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfileTwo, kSurfaceOperation)
    Call oExtrudeDef.SetDistanceExtent(2, kNegativeExtentDirection)
    Dim oExtrudeTwo As ExtrudeFeature
```

```
    oExtrudeTwo = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oSurfaces As ObjectCollection
    oSurfaces = ThisApplication.TransientObjects.CreateObjectCollection

    ' Since we know that the only work surfaces in the document
    ' are those created by the above extrusions, use those.
    oSurfaces.Add(oCompDef.WorkSurfaces.Item(1))
    oSurfaces.Add(oCompDef.WorkSurfaces.Item(2))

    ' Create a stitch (knit) feature by stitching
    ' together the two work surfaces created above.
    Dim oKnitFeature As KnitFeature
    oKnitFeature = oCompDef.Features.KnitFeatures.Add(oSurfaces)
```

Loft Feature With Non-Planar Section

Description

This sample demonstrates the creation of a loft feature. It uses two sections for the loft, one is from a 2d sketch and the other is a non-planar section from a 3d sketch. Because one of the sketches isn't planar, a surface is created instead of a solid.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateNonPlanarSectionLoft()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create a 2d sketch to use as one section.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))
    Call oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(0, 0), 5)

    ' Create the profile to use as the section.
    Dim oProfile1 As Profile
    Set oProfile1 = oSketch.Profiles.AddForSolid

    ' Create a 3d sketch to use as the second section.
    Dim oSketch3d As Sketch3D
    Set oSketch3d = oCompDef.Sketches3D.Add

    Dim oWPs(1 To 6) As WorkPoint
    Set oWPs(1) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(-8, 6, 10))
    Set oWPs(2) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(-8, -6, 10))
    Set oWPs(3) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, -4, 8))
    Set oWPs(4) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(8, -6, 10))
    Set oWPs(5) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(8, 6, 10))
    Set oWPs(6) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, 4, 8))
    Dim oStartLine3d As SketchLine3D
    Set oStartLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oWPs(1), oWPs(2), True, 2)
    Dim oLine3d As SketchLine3D
    Set oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oStartLine3d.EndSketchPoint, oWPs(3), True, 2)
    Set oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(4), True, 2)
    Set oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(5), True, 2)
    Set oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(6), True, 2)
    Set oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oStartLine3d.StartSketchPoint, True, 2)

    ' Create a 3d profile to use as the section. Even though this section
    ' is closed the AddOpen method must be used because it is non-planar.
    Dim oProfile2 As Profile3D
    Set oProfile2 = oSketch3d.Profiles3D.AddOpen

    ' Create an object collection for the sections.
    Dim oSections As ObjectCollection
    Set oSections = ThisApplication.TransientObjects.CreateObjectCollection
    Call oSections.Add(oProfile1)
    Call oSections.Add(oProfile2)

    ' Create the loft definition. Because one of the ends isn't planar,
    ' a surface must be created instead of a solid.
    Dim oLoftDefinition As LoftDefinition
    Set oLoftDefinition = oCompDef.Features.LoftFeatures.CreateLoftDefinition(oSections, kSurfaceOperation)

    ' Create the loft feature.
    Call oCompDef.Features.LoftFeatures.Add(oLoftDefinition)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
```

```
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create a 2d sketch to use as one section.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))
    Call oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(0, 0), 5)

    ' Create the profile to use as the section.
    Dim oProfile1 As Profile
    oProfile1 = oSketch.Profiles.AddForSolid

    ' Create a 3d sketch to use as the second section.
    Dim oSketch3d As Sketch3D
    oSketch3d = oCompDef.Sketches3D.Add

    Dim oWPs(0 To 5) As WorkPoint
    oWPs(0) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(-8, 6, 10))
    oWPs(1) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(-8, -6, 10))
    oWPs(2) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, -4, 8))
    oWPs(3) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(8, -6, 10))
    oWPs(4) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(8, 6, 10))
    oWPs(5) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, 4, 8))
    Dim oStartLine3d As SketchLine3D
    oStartLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oWPs(0), oWPs(1), True, 2)
    Dim oLine3d As SketchLine3D
    oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oStartLine3d.EndSketchPoint, oWPs(2), True, 2)
    oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(3), True, 2)
    oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(4), True, 2)
    oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oWPs(5), True, 2)
    oLine3d = oSketch3d.SketchLines3D.AddByTwoPoints(oLine3d.EndSketchPoint, oStartLine3d.StartSketchPoint, True, 2)

    ' Create a 3d profile to use as the section. Even though this section
    ' is closed the AddOpen method must be used because it is non-planar.
    Dim oProfile2 As Profile3D
    oProfile2 = oSketch3d.Profiles3D.AddOpen

    ' Create an object collection for the sections.
    Dim oSections As ObjectCollection
    oSections = ThisApplication.TransientObjects.CreateObjectCollection
    Call oSections.Add(oProfile1)
    Call oSections.Add(oProfile2)

    ' Create the loft definition. Because one of the ends isn't planar,
    ' a surface must be created instead of a solid.
    Dim oLoftDefinition As LoftDefinition
    oLoftDefinition = oCompDef.Features.LoftFeatures.CreateLoftDefinition(oSections, kSurfaceOperation)

    ' Create the loft feature.
    Call oCompDef.Features.LoftFeatures.Add(oLoftDefinition)
```

Mark feature creation sample

Description

This sample demonstrates how to create a mark feature in part document.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a mark feature in part document.

Copy Code

```
Sub MarkFeatureSample()
    ' Create a new part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
```

```
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    Set oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    Set oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    Set oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    Dim oGeometries As ObjectCollection
    Set oGeometries = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 1 To oRectangleLines.Count
        oGeometries.Add oRectangleLines(i)
    Next

    Dim oMarkFeatures As MarkFeatures
    Set oMarkFeatures = oCompDef.Features.MarkFeatures

    ' Get a mark style.
    Dim oMarkStyle As MarkStyle
    Set oMarkStyle = oPartDoc.MarkStyles.Item(1)

    ' Create mark definition.
    Dim oMarkDef As MarkDefinition
    Set oMarkDef = oMarkFeatures.CreateMarkDefinition(oGeometries, oMarkStyle)

    ' Create a mark feature.
    Dim oMark As MarkFeature
    Set oMark = oMarkFeatures.Add(oMarkDef)
End Sub
```

This sample demonstrates how to create a mark feature in part document.

Copy Code

```
    ' Create a new part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    Dim oGeometries As ObjectCollection
    oGeometries = ThisApplication.TransientObjects.CreateObjectCollection

    Dim i As Long
    For i = 1 To oRectangleLines.Count
        oGeometries.Add(oRectangleLines(i))
    Next

    Dim oMarkFeatures As MarkFeatures
    oMarkFeatures = oCompDef.Features.MarkFeatures
```

```
    ' Get a mark style.
    Dim oMarkStyle As MarkStyle
    oMarkStyle = oPartDoc.MarkStyles.Item(1)

    ' Create mark definition.
    Dim oMarkDef As MarkDefinition
    oMarkDef = oMarkFeatures.CreateMarkDefinition(oGeometries, oMarkStyle)

    ' Create a mark feature.
    Dim oMark As MarkFeature
    oMark = oMarkFeatures.Add(oMarkDef)
```

Move Feature Creation

Description

Demonstrates the creation of a Move feature.

Code Samples

- VBA
- iLogic

To use this sample a part must be active.

Copy Code

```
Public Sub MoveFeatureCreationSample()
    ' Get the active part document.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    If oDoc Is Nothing Then
        MsgBox "No part document!" & vbCrLf & "Please open a part with solids in it for this sample to run.", vbCritical, "Autodesk In
        Exit Sub
    End If

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    If oCompDef.SurfaceBodies.Count = 0 Then
        MsgBox "No solids to move!" & vbCrLf & "Please open a part with solids in it for this sample to run.", vbCritical, "Autodesk I
        Exit Sub
    End If

    Dim oBodies As ObjectCollection
    Set oBodies = ThisApplication.TransientObjects.CreateObjectCollection

    ' Specify a body to move.
    oBodies.Add oCompDef.SurfaceBodies(1)

    ' Create a MoveFeatureDefinition.
    Dim oMoveDef As MoveDefinition
    Set oMoveDef = oCompDef.Features.MoveFeatures.CreateMoveDefinition(oBodies)

    ' Set the move operations onto the bodies.
    Dim oFreeDrag As FreeDragMoveOperation
    Set oFreeDrag = oMoveDef.AddFreeDrag(1, 1, 1)

    Dim oMoveAlongRay As MoveAlongRayMoveOperation
    Set oMoveAlongRay = oMoveDef.AddMoveAlongRay(oCompDef.WorkAxes(2), True, 2)

    Dim oRotateAboutAxis As RotateAboutLineMoveOperation
    Set oRotateAboutAxis = oMoveDef.AddRotateAboutAxis(oCompDef.WorkAxes(3), True, 0.5)

    ' Create the move feature.
    Dim oMoveFeature As MoveFeature
    Set oMoveFeature = oCompDef.Features.MoveFeatures.Add(oMoveDef)
End Sub
```

To use this sample a part must be active.

Copy Code

```
    ' Get the active part document.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    If oDoc Is Nothing Then
        MsgBox("No part document!" & vbCrLf & "Please open a part with solids in it for this sample to run.", vbCritical, "Autodesk In
        Exit Sub
    End If

    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    If oCompDef.SurfaceBodies.Count = 0 Then
        MsgBox("No solids to move!" & vbCrLf & "Please open a part with solids in it for this sample to run.", vbCritical, "Autodesk I
        Exit Sub
    End If

    Dim oBodies As ObjectCollection
    oBodies = ThisApplication.TransientObjects.CreateObjectCollection

    ' Specify a body to move.
    oBodies.Add(oCompDef.SurfaceBodies(1))

    ' Create a MoveFeatureDefinition.
    Dim oMoveDef As MoveDefinition
```

```
    oMoveDef = oCompDef.Features.MoveFeatures.CreateMoveDefinition(oBodies)

    ' Set the move operations onto the bodies.
    Dim oFreeDrag As FreeDragMoveOperation
    oFreeDrag = oMoveDef.AddFreeDrag(1, 1, 1)

    Dim oMoveAlongRay As MoveAlongRayMoveOperation
    oMoveAlongRay = oMoveDef.AddMoveAlongRay(oCompDef.WorkAxes(2), True, 2)

    Dim oRotateAboutAxis As RotateAboutLineMoveOperation
    oRotateAboutAxis = oMoveDef.AddRotateAboutAxis(oCompDef.WorkAxes(3), True, 0.5)

    ' Create the move feature.
    Dim oMoveFeature As MoveFeature
    oMoveFeature = oCompDef.Features.MoveFeatures.Add(oMoveDef)
```

Create and Edit an Extrude Feature with a pocket

Description

This sample demonstrates how to edit an extrude feature. It shows how to create a sketch plane at a specified orientation to existing geometry.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub EditExtrudeFeature()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    Set oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    Set oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    Set oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    ' Create a profile.
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a pocket .25 cm deep (using distance extent).
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    Dim oExtrude2 As ExtrudeFeature
    Set oExtrude2 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Move the end of part above this extrude feature since the edit
    ' we are going to perform on this feature involves BREP input.
    Call oExtrude2.SetEndOfPart(True)

    ' Get the back face of the first extrusion to use as termination plane.
    Dim oBackFace As Face
    Set oBackFace = oExtrude1.EndFaces.Item(1)
```

```
    ' Change the extent type of the feature.
    Call oExtrude2.Definition.SetToExtent(oBackFace, False)

    ' Move the end of part back to bottom of the feature tree.
    Call oCompDef.SetEndOfPartToTopOrBottom(False)

    ' The following edit of the feature does not involve BREP input.
    ' Hence, no need to move the end of part.
    Call oExtrude2.Definition.SetDistanceExtent(0.25, kNegativeExtentDirection)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oTransGeom.CreatePoint2d(0, 0), _
    oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude1 As ExtrudeFeature
    oExtrude1 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Get the top face of the extrusion to use for creating the new sketch.
    Dim oFrontFace As Face
    oFrontFace = oExtrude1.StartFaces.Item(1)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (0.5,0.5,0) is.
    Dim oCorner As Point2d
    oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(0.5, 0.5, 0))

    ' Create the interior 3cm x 2cm rectangle for the pocket.
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
    oCorner, oTransGeom.CreatePoint2d(oCorner.X + 3, oCorner.Y + 2))

    ' Create a profile.
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a pocket .25 cm deep (using distance extent).
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kCutOperation)
    Call oExtrudeDef.SetDistanceExtent(0.25, kNegativeExtentDirection)
    Dim oExtrude2 As ExtrudeFeature
    oExtrude2 = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Move the end of part above this extrude feature since the edit
    ' we are going to perform on this feature involves BREP input.
    Call oExtrude2.SetEndOfPart(True)

    ' Get the back face of the first extrusion to use as termination plane.
    Dim oBackFace As Face
    oBackFace = oExtrude1.EndFaces.Item(1)

    ' Change the extent type of the feature.
    Call oExtrude2.Definition.SetToExtent(oBackFace, False)

    ' Move the end of part back to bottom of the feature tree.
    Call oCompDef.SetEndOfPartToTopOrBottom(False)

    ' The following edit of the feature does not involve BREP input.
    ' Hence, no need to move the end of part.
    Call oExtrude2.Definition.SetDistanceExtent(0.25, kNegativeExtentDirection)
```

Partial Chamfer Sample

Description

This sample demonstrates how to edit a chamfer feature to set the partial chamfer on a chamfered edge.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub PartialChamferSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oSk As PlanarSketch
    Set oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    Call oSk.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-5, -5), oTG.CreatePoint2d(5, 5))

    Dim oProfile As Profile
    Set oProfile = oSk.Profiles.AddForSolid

    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)

    Call oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oEdges As EdgeCollection
    Set oEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    oEdges.Add oDoc.ComponentDefinition.SurfaceBodies(1).Edges(1)

    Dim oChamfer As ChamferFeature
    Set oChamfer = oDoc.ComponentDefinition.Features.ChamferFeatures.AddUsingDistance(oEdges, 0.5, True)

    Dim oChamferDef As ChamferDefinition
    Set oChamferDef = oChamfer.Definition

    ' Rollback the End of Part node to be above the chamfer feature to edit it.
    Call oChamfer.SetEndOfPart(True)

    Dim oPartialChamferEdges As PartialChamferEdges
    Set oPartialChamferEdges = oChamferDef.PartialChamferEdges

    ' Set the partial chamfer edge.
    Call oPartialChamferEdges.Add(oDoc.ComponentDefinition.SurfaceBodies(1).Edges(1), 4, 3)

    ' After editing the feature, set it to be above the End of Part node.
    Call oChamfer.SetEndOfPart(False)
End Sub
```

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oSk As PlanarSketch
    oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    Call oSk.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-5, -5), oTG.CreatePoint2d(5, 5))

    Dim oProfile As Profile
    oProfile = oSk.Profiles.AddForSolid

    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oDoc.ComponentDefinition.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)

    Call oDoc.ComponentDefinition.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oEdges As EdgeCollection
    oEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    oEdges.Add(oDoc.ComponentDefinition.SurfaceBodies(1).Edges(1))

    Dim oChamfer As ChamferFeature
    oChamfer = oDoc.ComponentDefinition.Features.ChamferFeatures.AddUsingDistance(oEdges, 0.5, True)

    Dim oChamferDef As ChamferDefinition
    oChamferDef = oChamfer.Definition

    ' Rollback the End of Part node to be above the chamfer feature to edit it.
    Call oChamfer.SetEndOfPart(True)

    Dim oPartialChamferEdges As PartialChamferEdges
    oPartialChamferEdges = oChamferDef.PartialChamferEdges

    ' Set the partial chamfer edge.
    Call oPartialChamferEdges.Add(oDoc.ComponentDefinition.SurfaceBodies(1).Edges(1), 4, 3)

    ' After editing the feature, set it to be above the End of Part node.
    Call oChamfer.SetEndOfPart(False)
```

Part SimplifyFeature Sample

Description

This sample demonstrates how to create a simplify feature in part document.

Code Samples

- VBA
- iLogic

This sample demonstrates how to create a simplify feature in part document. You need to open a part document firstly before running below sample code.

Copy Code

```
Sub PartSimplifyFeatureSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oSimplifyFeatures As SimplifyFeatures
    Set oSimplifyFeatures = oCompDef.Features.SimplifyFeatures

    Dim oSimplifyDef As SimplifyDefinition
    Set oSimplifyDef = oSimplifyFeatures.CreateDefinition()

    ' Configure the options for simplify feature.
    oSimplifyDef.EnvelopesReplaceStyle = kEachBodyReplaceStyle
    oSimplifyDef.EnvelopeBoundingType = kOrientedMinimumBoundingBox
    oSimplifyDef.RemoveInternalBodies = False
    oSimplifyDef.RemoveBodiesBySize = True
    ' remove bodies which diagonal size is less than 10 centimeters.
    oSimplifyDef.RemoveBodiesSize = 10

    ' Create the Simplify feature.
    Dim oSimplifyFeature As SimplifyFeature
    Set oSimplifyFeature = oSimplifyFeatures.Add(oSimplifyDef)
End Sub
```

This sample demonstrates how to create a simplify feature in part document. You need to open a part document firstly before running below sample code.

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oSimplifyFeatures As SimplifyFeatures
    oSimplifyFeatures = oCompDef.Features.SimplifyFeatures

    Dim oSimplifyDef As SimplifyDefinition
    oSimplifyDef = oSimplifyFeatures.CreateDefinition()

    ' Configure the options for simplify feature.
    oSimplifyDef.EnvelopesReplaceStyle = kEachBodyReplaceStyle
    oSimplifyDef.EnvelopeBoundingType = kOrientedMinimumBoundingBox
    oSimplifyDef.RemoveInternalBodies = False
    oSimplifyDef.RemoveBodiesBySize = True
    ' remove bodies which diagonal size is less than 10 centimeters.
    oSimplifyDef.RemoveBodiesSize = 10

    ' Create the Simplify feature.
    Dim oSimplifyFeature As SimplifyFeature
    oSimplifyFeature = oSimplifyFeatures.Add(oSimplifyDef)
```

Sweep Feature Add

Description

This sample demonstrates the creation of a sweep feature. The profile is a circle, but the path is made up of a 3D sketch and a 2D sketch.

Code Samples

- VBA
- iLogic

To run the sample have a part document open.

Copy Code

```
Public Sub SweepFeature()
    ' Set a reference to the currently active document.
    ' This assumes that it is a part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition
```

```
    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create some workpoints that will be used for the 3D sketch.
    Dim oWPs(1 To 5) As WorkPoint
    Set oWPs(1) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, 0, 0))
    Set oWPs(2) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 0, 0))
    Set oWPs(3) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 4, 0))
    Set oWPs(4) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 4, 2))
    Set oWPs(5) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(6, 4, 2))

    ' Create a new 3D Sketch.
    Dim oSketch3D As Sketch3D
    Set oSketch3D = oPartDoc.ComponentDefinition.Sketches3D.Add

    ' Draw 3D lines. The first line is drawn between two of the work points.
    Dim oLine As SketchLine3D
    Set oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oWPs(1), oWPs(2), True, 1)

    ' This second and subsequent lines are drawn between a 3D sketch point and a work point.
    ' The work point is obtained from the previous line. Because the two lines will share
    ' this 3D sketch point, Inventor will treat them as connected lines when creating any
    ' paths.
    Set oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(3), True, 1)
    Set oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(4), True, 0.75)
    Set oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(5), True, 1)

    ' Create a 2D sketch.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(2))

    ' Determine the model origin relative to the sketch space.
    Dim oOrigin As Point2d
    Set oOrigin = oSketch.ModelToSketchSpace(oTG.CreatePoint(0, 0, 0))

    ' Create two lines.
    Dim oNewPoint As Point2d
    Set oNewPoint = oTG.CreatePoint2d(oOrigin.X, oOrigin.Y - 4)
    Dim oSketchLine1 As SketchLine
    Set oSketchLine1 = oSketch.SketchLines.AddByTwoPoints(oOrigin, oNewPoint)
    oNewPoint.X = oNewPoint.X + 3
    Dim oSketchLine2 As SketchLine
    Set oSketchLine2 = oSketch.SketchLines.AddByTwoPoints(oSketchLine1.EndSketchPoint, oNewPoint)

    ' Create a fillet between the two lines.
    Call oSketch.SketchArcs.AddByFillet(oSketchLine1, oSketchLine2, 1, oSketchLine1.StartSketchPoint.Geometry, oSketchLine2.EndSketchP

    ' Get the end of the 2d sketch in model space.
    Dim oModelPoint As Point
    Set oModelPoint = oSketch.SketchToModelSpace(oSketchLine2.EndSketchPoint.Geometry)

    ' Create a work plane at the end of the 2D sketch.
    Dim oWP As WorkPlane
    Set oWP = oCompDef.WorkPlanes.AddByNormalToCurve(oSketchLine2, oSketchLine2.EndSketchPoint)

    ' Create a path. Because the 3D and 2D sketches physically connect the path will include
    ' both of them.
    Dim oPath As Path
    Set oPath = oCompDef.Features.CreatePath(oSketchLine2)

    ' Create a sketch containing a circle.
    Set oSketch = oCompDef.Sketches.Add(oWP)
    Set oOrigin = oSketch.ModelToSketchSpace(oTG.CreatePoint(0, 0, 0))
    Call oSketch.SketchCircles.AddByCenterRadius(oSketch.ModelToSketchSpace(oModelPoint), 0.375)

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create the sweep feature.
    Dim oSweep As SweepFeature
    Set oSweep = oCompDef.Features.SweepFeatures.AddUsingPath(oProfile, oPath, kJoinOperation)
End Sub
```

To run the sample have a part document open.

Copy Code

```
    ' Set a reference to the currently active document.
    ' This assumes that it is a part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Set a reference to the transient geometry object.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create some workpoints that will be used for the 3D sketch.
    Dim oWPs(0 To 4) As WorkPoint
    oWPs(0) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(0, 0, 0))
    oWPs(1) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 0, 0))
    oWPs(2) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 4, 0))
    oWPs(3) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(3, 4, 2))
    oWPs(4) = oCompDef.WorkPoints.AddFixed(oTG.CreatePoint(6, 4, 2))

    ' Create a new 3D Sketch.
    Dim oSketch3D As Sketch3D
    oSketch3D = oPartDoc.ComponentDefinition.Sketches3D.Add

    ' Draw 3D lines. The first line is drawn between two of the work points.
    Dim oLine As SketchLine3D
    oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oWPs(0), oWPs(1), True, 1)
```

```
    ' This second and subsequent lines are drawn between a 3D sketch point and a work point.
    ' The work point is obtained from the previous line. Because the two lines will share
    ' this 3D sketch point, Inventor will treat them as connected lines when creating any
    ' paths.
    oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(2), True, 1)
    oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(3), True, 0.75)
    oLine = oSketch3D.SketchLines3D.AddByTwoPoints(oLine.EndPoint, oWPs(4), True, 1)

    ' Create a 2D sketch.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(2))

    ' Determine the model origin relative to the sketch space.
    Dim oOrigin As Point2d
    oOrigin = oSketch.ModelToSketchSpace(oTG.CreatePoint(0, 0, 0))

    ' Create two lines.
    Dim oNewPoint As Point2d
    oNewPoint = oTG.CreatePoint2d(oOrigin.X, oOrigin.Y - 4)
    Dim oSketchLine1 As SketchLine
    oSketchLine1 = oSketch.SketchLines.AddByTwoPoints(oOrigin, oNewPoint)
    oNewPoint.X = oNewPoint.X + 3
    Dim oSketchLine2 As SketchLine
    oSketchLine2 = oSketch.SketchLines.AddByTwoPoints(oSketchLine1.EndSketchPoint, oNewPoint)

    ' Create a fillet between the two lines.
    Call oSketch.SketchArcs.AddByFillet(oSketchLine1, oSketchLine2, 1, oSketchLine1.StartSketchPoint.Geometry, oSketchLine2.EndSketchP

    ' Get the end of the 2d sketch in model space.
    Dim oModelPoint As Point
    oModelPoint = oSketch.SketchToModelSpace(oSketchLine2.EndSketchPoint.Geometry)

    ' Create a work plane at the end of the 2D sketch.
    Dim oWP As WorkPlane
    oWP = oCompDef.WorkPlanes.AddByNormalToCurve(oSketchLine2, oSketchLine2.EndSketchPoint)

    ' Create a path. Because the 3D and 2D sketches physically connect the path will include
    ' both of them.
    Dim oPath As Path
    oPath = oCompDef.Features.CreatePath(oSketchLine2)

    ' Create a sketch containing a circle.
    oSketch = oCompDef.Sketches.Add(oWP)
    oOrigin = oSketch.ModelToSketchSpace(oTG.CreatePoint(0, 0, 0))
    Call oSketch.SketchCircles.AddByCenterRadius(oSketch.ModelToSketchSpace(oModelPoint), 0.375)

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create the sweep feature.
    Dim oSweep As SweepFeature
    oSweep = oCompDef.Features.SweepFeatures.AddUsingPath(oProfile, oPath, kJoinOperation)
```

Thread Feature Create

Description

This sample demonstrates the creation of a thread feature. It creates a cylinder in a new part document and creates a thread feature on the cylinder.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ThreadSample()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create a sketch circle.  (The radius is for 1 5/8 inch diameter shaft.)
    Dim oCircle As SketchCircle
    Set oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                  ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 4.1275 / 2)

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Extrude the circle to create a cylinder.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    oSketch.Visible = False
```

```
    ' Set a reference to the ThreadFeatures collection object.
    Dim oThreadFeatures As ThreadFeatures
    Set oThreadFeatures = oCompDef.Features.ThreadFeatures

    ' Define all of the thread information.
    Dim oThreadInfo As ThreadInfo
    Set oThreadInfo = oThreadFeatures.CreateStandardThreadInfo( _
                      False, True, "ANSI Unified Screw Threads", _
                      "1 5/8-6 UN", "2A")

    ' Get the face the thread will be applied to.
    Dim oFace As Face
    Set oFace = oExtrude.SideFaces.Item(1)

    ' Get the edge the thread extent will be measured from.
    Dim oEdge As Edge
    Set oEdge = oExtrude.EndFaces.Item(1).Edges.Item(1)

    ' Create the thread feature.
    Dim oThreadFeature As ThreadFeature
    Set oThreadFeature = oThreadFeatures.Add(oFace, oEdge, oThreadInfo, _
                                             False, False, "2 cm", 0)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Create a sketch circle.  (The radius is for 1 5/8 inch diameter shaft.)
    Dim oCircle As SketchCircle
    oCircle = oSketch.SketchCircles.AddByCenterRadius( _
                  ThisApplication.TransientGeometry.CreatePoint2d(0, 0), 4.1275 / 2)

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Extrude the circle to create a cylinder.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    oSketch.Visible = False

    ' Set a reference to the ThreadFeatures collection object.
    Dim oThreadFeatures As ThreadFeatures
    oThreadFeatures = oCompDef.Features.ThreadFeatures

    ' Define all of the thread information.
    Dim oThreadInfo As ThreadInfo
    oThreadInfo = oThreadFeatures.CreateStandardThreadInfo( _
                      False, True, "ANSI Unified Screw Threads", _
                      "1 5/8-6 UN", "2A")

    ' Get the face the thread will be applied to.
    Dim oFace As Face
    oFace = oExtrude.SideFaces.Item(1)

    ' Get the edge the thread extent will be measured from.
    Dim oEdge As Edge
    oEdge = oExtrude.EndFaces.Item(1).Edges.Item(1)

    ' Create the thread feature.
    Dim oThreadFeature As ThreadFeature
    oThreadFeature = oThreadFeatures.Add(oFace, oEdge, oThreadInfo, _
                                         False, False, "2 cm", 0)
```

Edit thread features

Description

The following example demonstrates how to edit an existing thread feature.

Code Samples

- VBA
- iLogic

You need to make sure that the ThreadInfo object that you create is appropriate (size, etc.) for the face that it will be applied for. A ThreadInfo object represents a row in the Thread.xls spreadsheet.

Copy Code

```
Sub EditThread()
   Dim oDoc As PartDocument
```

```
    Set oDoc = ThisApplication.ActiveDocument

    Dim oDef As PartComponentDefinition
    Set oDef = oDoc.ComponentDefinition

    ' Create a new thread info object containing the thread data
    Dim oNewThreadInfo As StandardThreadInfo
    Set oNewThreadInfo = oDef.Features.ThreadFeatures.CreateStandardThreadInfo(False, True, "ISO Metric Profile", "M20x2.5", "6g")

    ' Get the first thread feature
    Dim oThread As ThreadFeature
    Set oThread = oDef.Features.ThreadFeatures.Item(1)

    ' Edit the thread feature
    oThread.ThreadInfo = oNewThreadInfo
End Sub
```

You need to make sure that the ThreadInfo object that you create is appropriate (size, etc.) for the face that it will be applied for. A ThreadInfo object represents a row in the Thread.xls spreadsheet.

Copy Code

```
     Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oDef As PartComponentDefinition
    oDef = oDoc.ComponentDefinition

    ' Create a new thread info object containing the thread data
    Dim oNewThreadInfo As StandardThreadInfo
    oNewThreadInfo = oDef.Features.ThreadFeatures.CreateStandardThreadInfo(False, True, "ISO Metric Profile", "M20x2.5", "6g")

    ' Get the first thread feature
    Dim oThread As ThreadFeature
    oThread = oDef.Features.ThreadFeatures.Item(1)

    ' Edit the thread feature
    oThread.ThreadInfo = oNewThreadInfo
```

Creating a ThreadInfo object

Description

Demonstrates the use of a ThreadInfo object.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub test_threadinfo()
    Dim oApp As Application
    Set oApp = ThisApplication

    Dim oGeneralOptions As GeneralOptions
    Set oGeneralOptions = oApp.GeneralOptions

    Dim oThreadTable As ThreadTableQuery
    Set oThreadTable = oGeneralOptions.ThreadTableQuery

    Dim oTypes() As String
    oTypes = oThreadTable.GetAvailableThreadTypes

    Dim oSizes() As String
    oSizes = oThreadTable.GetAvailableThreadSizes(False, "NPT")

    Dim oDesignations() As String
    oDesignations = oThreadTable.GetAvailableDesignations(False, "NPT", oSizes(1))

    Dim oThreadInfo As TaperedThreadInfo
    Set oThreadInfo = oThreadTable.CreateThreadInfo(False, True, "NPT", oDesignations(0))
End Sub
```

Copy Code

```
    Dim oApp As Application
    oApp = ThisApplication

    Dim oGeneralOptions As GeneralOptions
    oGeneralOptions = oApp.GeneralOptions

    Dim oThreadTable As ThreadTableQuery
    oThreadTable = oGeneralOptions.ThreadTableQuery

    Dim oTypes() As String
    oTypes = oThreadTable.GetAvailableThreadTypes

    Dim oSizes() As String
    oSizes = oThreadTable.GetAvailableThreadSizes(False, "NPT")

    Dim oDesignations() As String
    oDesignations = oThreadTable.GetAvailableDesignations(False, "NPT", oSizes(1))

    Dim oThreadInfo As TaperedThreadInfo
    oThreadInfo = oThreadTable.CreateThreadInfo(False, True, "NPT", oDesignations(0))
```

Creating a new parameter group

Description

This sample demonstrates the creation of model, reference and user parameters and copying these parameters to a newly created group.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateParametersAndGroup()
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                   ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a model parameter
    Dim oModelParam As ModelParameter
    Set oModelParam = oCompDef.Parameters.ModelParameters.AddByValue(2, kCentimeterLengthUnits)

    ' Create a reference parameter
    Dim oReferenceParam As ReferenceParameter
    Set oReferenceParam = oCompDef.Parameters.ReferenceParameters.AddByValue(4, kCentimeterLengthUnits)

    ' Create a user parameter
    Dim oUserParam As UserParameter
    Set oUserParam = oCompDef.Parameters.UserParameters.AddByValue("length", 6, kCentimeterLengthUnits)

    ' Create a new custom parameter group
    Dim oCustomParamGroup As CustomParameterGroup
    Set oCustomParamGroup = oCompDef.Parameters.CustomParameterGroups.Add("Custom Group", "CustomGroup1")

    ' Add the created parameters to this group
    ' Note that adding the parameters to the custom group
    ' does not remove it from the original group.
    Call oCustomParamGroup.Add(oModelParam)
    Call oCustomParamGroup.Add(oReferenceParam)
    Call oCustomParamGroup.Add(oUserParam)
End Sub
```

Copy Code

```
    ' Create a new Part document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                   ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the compdef.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a model parameter
    Dim oModelParam As ModelParameter
    oModelParam = oCompDef.Parameters.ModelParameters.AddByValue(2, kCentimeterLengthUnits)

    ' Create a reference parameter
    Dim oReferenceParam As ReferenceParameter
    oReferenceParam = oCompDef.Parameters.ReferenceParameters.AddByValue(4, kCentimeterLengthUnits)

    ' Create a user parameter
    Dim oUserParam As UserParameter
    oUserParam = oCompDef.Parameters.UserParameters.AddByValue("length", 6, kCentimeterLengthUnits)

    ' Create a new custom parameter group
    Dim oCustomParamGroup As CustomParameterGroup
    oCustomParamGroup = oCompDef.Parameters.CustomParameterGroups.Add("Custom Group", "CustomGroup1")

    ' Add the created parameters to this group
    ' Note that adding the parameters to the custom group
    ' does not remove it from the original group.
    Call oCustomParamGroup.Add(oModelParam)
    Call oCustomParamGroup.Add(oReferenceParam)
    Call oCustomParamGroup.Add(oUserParam)
```

Derived Parts and Assemblies

Description

This sample demonstrates the use of the API to create derived parts and assemblies.

Code Samples

- VBA
- iLogic

This example used is a simple case of a part subtracted from a mold. To simplify setup, the program creates the parts representing the part and the mold. It then uses derived parts and assemblies to scale the part and subtract it from the mold.

Copy Code

```
Public Sub MoldBaseSample()
    ' Initialize the string that defines the directory where the
    ' files will be created.
    Dim sFilePath As String
    sFilePath = "C:\Temp\"

    ' Call the functions to create the parts that represent the
    ' molded part and the mold base.
    Call CreateMoldPart(sFilePath & "MoldPart.ipt")
    Call CreateMoldBase(sFilePath & "MoldBase.ipt")

    ' Create a new part file to derive the mold part in.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a derived definition for the molded part.
    Dim oDerivedPartDef As DerivedPartUniformScaleDef
    Set oDerivedPartDef = oPartDoc.ComponentDefinition.ReferenceComponents.DerivedPartComponents.CreateUniformScaleDef(sFilePath & "Mo

    ' Set the scale to use.
    oDerivedPartDef.ScaleFactor = 1.1

    ' We could set other options for the derived part using the derived part definition.
    ' In this case the defaults are good except for the scale which we changed.

    ' Create the derived part.
    Call oPartDoc.ComponentDefinition.ReferenceComponents.DerivedPartComponents.Add(oDerivedPartDef)

    ' Save and close the part.  Uses SilentOperation to bypass the save dialog.
    ThisApplication.SilentOperation = True
    Call oPartDoc.SaveAs(sFilePath & "ScaledMoldPart.ipt", False)
    ThisApplication.SilentOperation = False
    oPartDoc.Close

    ' Create a new assembly file to put the mold base and scaled part together.
    Dim oAsmDoc As AssemblyDocument
    Set oAsmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))

    ' Create the matrix used to define the position of the occurrences.  When a new
    ' matrix is created it is initialized to an identity matrix.  This will cause
    ' parts to be placed into the assembly in the same location that are in part space.
    Dim oMatrix As Matrix
    Set oMatrix = ThisApplication.TransientGeometry.CreateMatrix

    ' Place the mold base.
    Dim oOcc As ComponentOccurrence
    Set oOcc = oAsmDoc.ComponentDefinition.Occurrences.Add(sFilePath & "MoldBase.ipt", oMatrix)

    ' Rename the occurrence.  This sample uses the name to identify the occurrence later.
    ' This isn't the only method that could be used, but is one of the simplest.
    oOcc.Name = "Mold Base"

    ' Place the scaled part.
    Set oOcc = oAsmDoc.ComponentDefinition.Occurrences.Add(sFilePath & "ScaledMoldPart.ipt", oMatrix)
    oOcc.Name = "Mold Part"

    ' Save and close the assembly.
    Call oAsmDoc.SaveAs(sFilePath & "MoldSample.iam", False)
    oAsmDoc.Close

    ' Create a new part to derive the assembly into, subtracting the part from the mold base.
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a derived definition for the molded assembly.
    Dim oDerivedAsmDef As DerivedAssemblyDefinition
    Set oDerivedAsmDef = oPartDoc.ComponentDefinition.ReferenceComponents.DerivedAssemblyComponents.CreateDefinition(sFilePath & "Mold

    ' Set the part to be subtracted.
    oDerivedAsmDef.Occurrences.Item("Mold Part").InclusionOption = kDerivedSubtractAll

    ' Create the derived assembly.
    Call oPartDoc.ComponentDefinition.ReferenceComponents.DerivedAssemblyComponents.Add(oDerivedAsmDef)
End Sub

' Sub to create the part representing the molded part.
Private Sub CreateMoldPart(Filename As String)
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a new sketch on the XY base workplane.
    Dim oSketch As PlanarSketch
    Set oSketch = oPartDoc.ComponentDefinition.Sketches.Add(oPartDoc.ComponentDefinition.WorkPlanes(3))

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Draw the geometry defining the shape of the part.
    Dim oPoints As ObjectCollection
    Set oPoints = ThisApplication.TransientObjects.CreateObjectCollection
    oPoints.Add oTG.CreatePoint2d(-5, 0)
    oPoints.Add oTG.CreatePoint2d(-4, 3)
```

```
    oPoints.Add oTG.CreatePoint2d(-2, 4)
    oPoints.Add oTG.CreatePoint2d(0, 3)
    oPoints.Add oTG.CreatePoint2d(3, 4)
    oPoints.Add oTG.CreatePoint2d(4, 2)
    oPoints.Add oTG.CreatePoint2d(5, 0)

    Dim oSpline As SketchSpline
    Set oSpline = oSketch.SketchSplines.Add(oPoints)
    oSpline.FitMethod = kSweetSplineFit

    Dim oLine As SketchLine
    Set oLine = oSketch.SketchLines.AddByTwoPoints(oSpline.FitPoint(1), oSpline.FitPoint(oSpline.FitPointCount))

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a revolved feature.
    Call oPartDoc.ComponentDefinition.Features.RevolveFeatures.AddFull(oProfile, oLine, kJoinOperation)

    ' Save and close the document.
    Call oPartDoc.SaveAs(Filename, False)
    oPartDoc.Close
End Sub

' Sub to create the part representing the mold base.
Private Sub CreateMoldBase(Filename As String)
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a new sketch on the XY base workplane.
    Dim oSketch As PlanarSketch
    Set oSketch = oPartDoc.ComponentDefinition.Sketches.Add(oPartDoc.ComponentDefinition.WorkPlanes(3))

    ' Draw the geometry defining the shape of the part.
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-6, -5), oTG.CreatePoint2d(6, 5))

    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extruded feature.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kNegativeExtentDirection)
    Call oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Save and close the document.
    Call oPartDoc.SaveAs(Filename, False)
    oPartDoc.Close
End Sub
```

This example used is a simple case of a part subtracted from a mold. To simplify setup, the program creates the parts representing the part and the mold. It then uses derived parts and assemblies to scale the part and subtract it from the mold.

Copy Code

```
Sub Main
    ' Initialize the string that defines the directory where the
    ' files will be created.
    Dim sFilePath As String
    sFilePath = "C:\Temp\"

    ' Call the functions to create the parts that represent the
    ' molded part and the mold base.
    Call CreateMoldPart(sFilePath & "MoldPart.ipt")
    Call CreateMoldBase(sFilePath & "MoldBase.ipt")

    ' Create a new part file to derive the mold part in.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a derived definition for the molded part.
    Dim oDerivedPartDef As DerivedPartUniformScaleDef
    oDerivedPartDef = oPartDoc.ComponentDefinition.ReferenceComponents.DerivedPartComponents.CreateUniformScaleDef(sFilePath & "MoldPa

    ' Set the scale to use.
    oDerivedPartDef.ScaleFactor = 1.1

    ' We could set other options for the derived part using the derived part definition.
    ' In this case the defaults are good except for the scale which we changed.

    ' Create the derived part.
    Call oPartDoc.ComponentDefinition.ReferenceComponents.DerivedPartComponents.Add(oDerivedPartDef)

    ' Save and close the part.  Uses SilentOperation to bypass the save dialog.
    ThisApplication.SilentOperation = True
    Call oPartDoc.SaveAs(sFilePath & "ScaledMoldPart.ipt", False)
    ThisApplication.SilentOperation = False
    oPartDoc.Close

    ' Create a new assembly file to put the mold base and scaled part together.
    Dim oAsmDoc As AssemblyDocument
    oAsmDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kAssemblyDocumentObject))

    ' Create the matrix used to define the position of the occurrences.  When a new
    ' matrix is created it is initialized to an identity matrix.  This will cause
    ' parts to be placed into the assembly in the same location that are in part space.
    Dim oMatrix As Matrix
    oMatrix = ThisApplication.TransientGeometry.CreateMatrix
```

```
    ' Place the mold base.
    Dim oOcc As ComponentOccurrence
    oOcc = oAsmDoc.ComponentDefinition.Occurrences.Add(sFilePath & "MoldBase.ipt", oMatrix)

    ' Rename the occurrence.  This sample uses the name to identify the occurrence later.
    ' This isn't the only method that could be used, but is one of the simplest.
    oOcc.Name = "Mold Base"

    ' Place the scaled part.
    oOcc = oAsmDoc.ComponentDefinition.Occurrences.Add(sFilePath & "ScaledMoldPart.ipt", oMatrix)
    oOcc.Name = "Mold Part"

    ' Save and close the assembly.
    Call oAsmDoc.SaveAs(sFilePath & "MoldSample.iam", False)
    oAsmDoc.Close

    ' Create a new part to derive the assembly into, subtracting the part from the mold base.
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a derived definition for the molded assembly.
    Dim oDerivedAsmDef As DerivedAssemblyDefinition
    oDerivedAsmDef = oPartDoc.ComponentDefinition.ReferenceComponents.DerivedAssemblyComponents.CreateDefinition(sFilePath & "MoldSamp

    ' Set the part to be subtracted.
    oDerivedAsmDef.Occurrences.Item("Mold Part").InclusionOption = kDerivedSubtractAll

    ' Create the derived assembly.
    Call oPartDoc.ComponentDefinition.ReferenceComponents.DerivedAssemblyComponents.Add(oDerivedAsmDef)
End Sub

' Sub to create the part representing the molded part.
Private Sub CreateMoldPart(Filename As String)
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a new sketch on the XY base workplane.
    Dim oSketch As PlanarSketch
    oSketch = oPartDoc.ComponentDefinition.Sketches.Add(oPartDoc.ComponentDefinition.WorkPlanes(3))

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Draw the geometry defining the shape of the part.
    Dim oPoints As ObjectCollection
    oPoints = ThisApplication.TransientObjects.CreateObjectCollection
    oPoints.Add(oTG.CreatePoint2d(-5, 0))
    oPoints.Add(oTG.CreatePoint2d(-4, 3))
    oPoints.Add(oTG.CreatePoint2d(-2, 4))
    oPoints.Add(oTG.CreatePoint2d(0, 3))
    oPoints.Add(oTG.CreatePoint2d(3, 4))
    oPoints.Add(oTG.CreatePoint2d(4, 2))
    oPoints.Add(oTG.CreatePoint2d(5, 0))

    Dim oSpline As SketchSpline
    oSpline = oSketch.SketchSplines.Add(oPoints)
    oSpline.FitMethod = kSweetSplineFit

    Dim oLine As SketchLine
    oLine = oSketch.SketchLines.AddByTwoPoints(oSpline.FitPoint(1), oSpline.FitPoint(oSpline.FitPointCount))

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a revolved feature.
    Call oPartDoc.ComponentDefinition.Features.RevolveFeatures.AddFull(oProfile, oLine, kJoinOperation)

    ' Save and close the document.
    Call oPartDoc.SaveAs(Filename, False)
    oPartDoc.Close
End Sub

' Sub to create the part representing the mold base.
Private Sub CreateMoldBase(Filename As String)
    ' Create a new part document using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Create a new sketch on the XY base workplane.
    Dim oSketch As PlanarSketch
    oSketch = oPartDoc.ComponentDefinition.Sketches.Add(oPartDoc.ComponentDefinition.WorkPlanes(3))

    ' Draw the geometry defining the shape of the part.
    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry
    Call oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(-6, -5), oTG.CreatePoint2d(6, 5))

    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extruded feature.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kJoinOperation)
    Call oExtrudeDef.SetDistanceExtent(5, kNegativeExtentDirection)
    Call oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Save and close the document.
    Call oPartDoc.SaveAs(Filename, False)
    oPartDoc.Close
End Sub
```

Selectively link paramaters

Description

This sample demonstrates the selective linking of parameters from another Inventor file.

Code Samples

- VBA
- iLogic

Before running this sample, make sure that the file C:\temp\block.ipt exists (or change the path in the sample). The sample assumes that the file contains parameters named d0, d1 and d2.

Copy Code

```
Sub SelectivelyLinkParams()
    ' Open the source document invisible.
    Dim oSourceDoc As PartDocument
    Set oSourceDoc = ThisApplication.Documents.Open("C:\temp\block.ipt", False)

    ' Set a reference to the component definition.
    Dim oSourceCompDef As PartComponentDefinition
    Set oSourceCompDef = oSourceDoc.ComponentDefinition

    Dim oParamsToLink As ObjectCollection
    Set oParamsToLink = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add parameters named "d0" and "d1".
    ' This assumes that the source document contains
    ' parameters with these names.
    oParamsToLink.Add oSourceCompDef.Parameters.Item("d0")
    oParamsToLink.Add oSourceCompDef.Parameters.Item("d1")

    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Create a derived parameter table that links only to "d0"
    ' and "d1" in the source part.
    ' Note: If parameters "d0" and "d1" in the source part
    ' are not already exported, they will be automatically
    ' exported and hence will result in changing the source part.
    Dim oDerivedParamTable As DerivedParameterTable
    Set oDerivedParamTable = oPartDoc.ComponentDefinition.Parameters. _
        DerivedParameterTables.Add2("C:\temp\block.ipt", oParamsToLink)

    ' Add parameter named "d2"
    ' This assumes that the source document
    ' contains a parameters named "d2".
    oParamsToLink.Add oSourceCompDef.Parameters.Item("d2")

    ' Change derived parameter table so it also links to "d2".
    oDerivedParamTable.LinkedParameters = oParamsToLink
End Sub
```

Before running this sample, make sure that the file C:\temp\block.ipt exists (or change the path in the sample). The sample assumes that the file contains parameters named d0, d1 and d2.

Copy Code

```
    ' Open the source document invisible.
    Dim oSourceDoc As PartDocument
    oSourceDoc = ThisApplication.Documents.Open("C:\temp\block.ipt", False)

    ' Set a reference to the component definition.
    Dim oSourceCompDef As PartComponentDefinition
    oSourceCompDef = oSourceDoc.ComponentDefinition

    Dim oParamsToLink As ObjectCollection
    oParamsToLink = ThisApplication.TransientObjects.CreateObjectCollection

    ' Add parameters named "d0" and "d1".
    ' This assumes that the source document contains
    ' parameters with these names.
    oParamsToLink.Add(oSourceCompDef.Parameters.Item("d0"))
    oParamsToLink.Add(oSourceCompDef.Parameters.Item("d1"))

    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Create a derived parameter table that links only to "d0"
    ' and "d1" in the source part.
    ' Note: If parameters "d0" and "d1" in the source part
    ' are not already exported, they will be automatically
    ' exported and hence will result in changing the source part.
    Dim oDerivedParamTable As DerivedParameterTable
    oDerivedParamTable = oPartDoc.ComponentDefinition.Parameters. _
        DerivedParameterTables.Add2("C:\temp\block.ipt", oParamsToLink)

    ' Add parameter named "d2"
    ' This assumes that the source document
    ' contains a parameters named "d2".
    oParamsToLink.Add(oSourceCompDef.Parameters.Item("d2"))

    ' Change derived parameter table so it also links to "d2".
    oDerivedParamTable.LinkedParameters = oParamsToLink
```

Query feature dimensions

Description

This sample demonstrates querying the dimensions of a feature in a part or an assembly document. The sample also demonstrates how to show the dimensions of a feature.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub QueryAndShowFeatureDimensions()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a feature is selected.
    If Not Typeof oDoc.SelectSet.Item(1) Is PartFeature Then
        MsgBox "A feature must be selected."
        Exit Sub
    End If

    ' Set a reference to the selected feature.
    Dim oFeature As PartFeature
    Set oFeature = oDoc.SelectSet.Item(1)

    Dim oFeatureDim As FeatureDimension
    Dim i As Long
    i = 0
    ' Iterate over all dimensions of the feature.
    For Each oFeatureDim In oFeature.FeatureDimensions
        i = i + 1
        ' Format and print the name and position.
        Debug.Print i & ". Name: " & oFeatureDim.Parameter.Name _
                 & "  Position: (" & oFeatureDim.TextPoint.X & _
                              ", " & oFeatureDim.TextPoint.Y & _
                              ", " & oFeatureDim.TextPoint.Z & ")"
    Next

    ' Show all the feature dimensions
    oFeature.FeatureDimensions.Show
End Sub
```

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    ' Check to make sure a feature is selected.
    If Not Typeof oDoc.SelectSet.Item(1) Is PartFeature Then
        MsgBox("A feature must be selected.")
        Exit Sub
    End If

    ' Set a reference to the selected feature.
    Dim oFeature As PartFeature
    oFeature = oDoc.SelectSet.Item(1)

    Dim oFeatureDim As FeatureDimension
    Dim i As Long
    i = 0
    ' Iterate over all dimensions of the feature.
    For Each oFeatureDim In oFeature.FeatureDimensions
        i = i + 1
        ' Format and print the name and position.
        Logger.Info(i & ". Name: " & oFeatureDim.Parameter.Name _
                 & "  Position: (" & oFeatureDim.TextPoint.X & _
                              ", " & oFeatureDim.TextPoint.Y & _
                              ", " & oFeatureDim.TextPoint.Z & ")")
    Next

    ' Show all the feature dimensions
    oFeature.FeatureDimensions.Show
```

Associatively import AutoCAD

Description

This sample demonstrates how to import AutoCAD associatively.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub AssociativelyImportAutoCADDWGSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oPartCompDef As PartComponentDefinition
    Set oPartCompDef = oDoc.ComponentDefinition

    ' Create the ImportedDWGComponentDefinition bases on an AutoCAD file
    Dim oImportedDWGCompDef As ImportedDWGComponentDefinition
    Set oImportedDWGCompDef = oPartCompDef.ReferenceComponents.ImportedComponents.CreateDefinition("C:\Temp\MyDesign.dwg")

    ' Import the AutoCAD DWG
    Dim oImportedComp As ImportedComponent
    Set oImportedComp = oPartCompDef.ReferenceComponents.ImportedComponents.Add(oImportedDWGCompDef)
End Sub
```

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oPartCompDef As PartComponentDefinition
    oPartCompDef = oDoc.ComponentDefinition

    ' Create the ImportedDWGComponentDefinition bases on an AutoCAD file
    Dim oImportedDWGCompDef As ImportedDWGComponentDefinition
    oImportedDWGCompDef = oPartCompDef.ReferenceComponents.ImportedComponents.CreateDefinition("C:\Temp\MyDesign.dwg")

    ' Import the AutoCAD DWG
    Dim oImportedComp As ImportedComponent
    oImportedComp = oPartCompDef.ReferenceComponents.ImportedComponents.Add(oImportedDWGCompDef)
```

Basic Selection Using Interaction Events

Description

This sample demonstrates using the selection events to select a face. Selection is dependent on events and VB only supports events within a class module.

Code Samples

- VBA
- iLogic

To use the sample copy the TestSelection sub into a code module. Create a new class module called clsSelect and copy all of the rest of the code into it. To run the sample, have a part document open that contains some geometry and run the TestSelection sub. Select a face and it will display its area.

Copy Code

```
'**********************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsSelect". The name can be
' changed but you'll need to change the declaration in the
' calling function "TestSelection" to use the new name.

' Declare the event objects
Private WithEvents oInteractEvents As InteractionEvents
Private WithEvents oSelectEvents As SelectEvents

' Declare a flag that's used to determine when selection stops.
Private bStillSelecting As Boolean

Public Function Pick(filter As SelectionFilterEnum) As Object
    ' Initialize flag.
    bStillSelecting = True

    ' Create an InteractionEvents object.
    Set oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

    ' Ensure interaction is enabled.
    oInteractEvents.InteractionDisabled = False

    ' Set a reference to the select events.
    Set oSelectEvents = oInteractEvents.SelectEvents

    ' Set the filter using the value passed in.
    oSelectEvents.AddSelectionFilter filter

    ' Start the InteractionEvents object.
    oInteractEvents.Start

    ' Loop until a selection is made.
    Do While bStillSelecting
        ThisApplication.UserInterfaceManager.DoEvents
    Loop

    ' Get the selected item. If more than one thing was selected,
    ' just get the first item and ignore the rest.
    Dim oSelectedEnts As ObjectsEnumerator
    Set oSelectedEnts = oSelectEvents.SelectedEntities
    If oSelectedEnts.Count > 0 Then
        Set Pick = oSelectedEnts.Item(1)
    Else
        Set Pick = Nothing
    End If
```

```
    ' Stop the InteractionEvents object.
    oInteractEvents.Stop

    ' Clean up.
    Set oSelectEvents = Nothing
    Set oInteractEvents = Nothing
End Function

Private Sub oInteractEvents_OnTerminate()
    ' Set the flag to indicate we're done.
    bStillSelecting = False
End Sub

Private Sub oSelectEvents_OnSelect(ByVal JustSelectedEntities As ObjectsEnumerator, ByVal SelectionDevice As SelectionDeviceEnum, ByVa
    Dim oFace As Face
    Set oFace = JustSelectedEntities(1)

    ' Check to make sure an object was selected.
    If Not oFace Is Nothing Then
        ' Display the area of the selected face.
        MsgBox "Face area: " & oFace.Evaluator.Area & " cm^2"
    End If
End Sub
```

To run the sample, have a part document open that contains some geometry and run the code. Select a face and it will display its area.

Copy Code

```
Sub Main
    ' Create a new clsSelect object.
    Dim oSelect As New clsSelect
        oSelect.Init(ThisApplication)

    ' Call the pick method of the clsSelect object and set
    ' the filter to pick any face.
    Dim oFace As Face
    oFace = oSelect.Pick(kPartFaceFilter)
End Sub

Class clsSelect
        ' Declare the event objects
        Private WithEvents oInteractEvents As InteractionEvents
        Private WithEvents oSelectEvents As SelectEvents
        Private ThisApplication As Inventor.Application

        ' Declare a flag that's used to determine when selection stops.
        Private bStillSelecting As Boolean

        Public Sub Init(oApp As Inventor.Application)
                ThisApplication = oApp
        End Sub

        Public Function Pick(filter As SelectionFilterEnum) As Object
            ' Initialize flag.
            bStillSelecting = True

            ' Create an InteractionEvents object.
            oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

            ' Ensure interaction is enabled.
            oInteractEvents.InteractionDisabled = False

            ' Set a reference to the select events.
            oSelectEvents = oInteractEvents.SelectEvents

            ' Set the filter using the value passed in.
            oSelectEvents.AddSelectionFilter(filter)

            ' Start the InteractionEvents object.
            oInteractEvents.Start

            ' Loop until a selection is made.
            Do While bStillSelecting
                ThisApplication.UserInterfaceManager.DoEvents
            Loop

            ' Get the selected item. If more than one thing was selected,
            ' just get the first item and ignore the rest.
            Dim oSelectedEnts As ObjectsEnumerator
            oSelectedEnts = oSelectEvents.SelectedEntities
            If oSelectedEnts.Count > 0 Then
                Pick = oSelectedEnts.Item(1)
            Else
                Pick = Nothing
            End If

            ' Stop the InteractionEvents object.
            oInteractEvents.Stop

            ' Clean up.
            oSelectEvents = Nothing
            oInteractEvents = Nothing
        End Function

        Private Sub oInteractEvents_OnTerminate() Handles oInteractEvents.OnTerminate
            ' Set the flag to indicate we're done.
            bStillSelecting = False
        End Sub

        Private Sub oSelectEvents_OnSelect(ByVal JustSelectedEntities As ObjectsEnumerator, ByVal SelectionDevice As SelectionDeviceEn
                Dim oFace As Face
        oFace = JustSelectedEntities(1) 'oSelect.Pick(kPartFaceFilter)

            ' Check to make sure an object was selected.
            If Not oFace Is Nothing Then
                ' Display the area of the selected face.
                MsgBox("Face area: " & oFace.Evaluator.Area & " cm^2")
            End If
```

```
        End Sub
End Class
```

Is cylindrical face interior or exterior?

Description

This sample shows how to determine whether the selected cylindircal face is an exterior face or an interior (hollow) face.

Code Samples

- VBA
- iLogic

Before running the sample, select a cylindrical face.

Copy Code

```
Public Sub IsCylindricalFaceInterior()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    If Not Typeof oDoc.SelectSet(1) Is Face Then
        MsgBox "A face must be selected."
        Exit Sub
    End If

    Dim oFace As Face
    Set oFace = oDoc.SelectSet(1)

    If Not oFace.SurfaceType = kCylinderSurface Then
        MsgBox "A cylindrical face must be selected."
        Exit Sub
    End If

    Dim oCylinder As Cylinder
    Set oCylinder = oFace.Geometry

    Dim params(1) As Double
    params(0) = 0.5
    params(1) = 0.5

    ' Get point on surface at param .5,.5
    Dim points(2) As Double
    Call oFace.Evaluator.GetPointAtParam(params, points)

    ' Create point object
    Dim oPoint As point
    Set oPoint = ThisApplication.TransientGeometry.CreatePoint(points(0), points(1), points(2))

    ' Get normal at this point
    Dim normals(2) As Double
    Call oFace.Evaluator.GetNormal(params, normals)

    ' Create normal vector object
    Dim oNormal As Vector
    Set oNormal = ThisApplication.TransientGeometry.CreateVector(normals(0), normals(1), normals(2))

    ' Scale vector by radius of the cylinder
    oNormal.ScaleBy oCylinder.Radius

    ' Find the sampler point on the normal by adding the
    ' scaled normal vector to the point at .5,.5 param.
    Dim oSamplePoint As point
    Set oSamplePoint = oPoint

    oSamplePoint.TranslateBy oNormal

    ' Check if the sample point lies on the cylinder axis.
    ' If it does, we have a hollow face.

    ' Create a line describing the cylinder axis
    Dim oAxisLine As Line
    Set oAxisLine = ThisApplication.TransientGeometry.CreateLine _
        (oCylinder.BasePoint, oCylinder.AxisVector.AsVector)

    'Create a line parallel to the axis passing thru the sample point.
    Dim oSampleLine As Line
    Set oSampleLine = ThisApplication.TransientGeometry.CreateLine _
        (oSamplePoint, oCylinder.AxisVector.AsVector)

    If oSampleLine.IsColinearTo(oAxisLine) Then
        MsgBox "Interior face."
    Else
        MsgBox "Exterior face."
    End If
End Sub
```

Before running the sample, select a cylindrical face.

Copy Code

```
    Dim oDoc As Document
    oDoc = ThisApplication.ActiveDocument

    If oDoc.SelectSet.Count = 0 Then
        MsgBox("A face must be selected.")
        Exit Sub
    Else
```

```
        If oDoc.SelectSet.Item(1).Type <> kFaceObject Then
            MsgBox("A face must be selected.")
            Exit Sub
        End If
    End If

    Dim oFace As Face
    oFace = oDoc.SelectSet(1)

    If Not oFace.SurfaceType = kCylinderSurface Then
        MsgBox("A cylindrical face must be selected.")
        Exit Sub
    End If

    Dim oCylinder As Cylinder
    oCylinder = oFace.Geometry

    Dim params(1) As Double
    params(0) = 0.5
    params(1) = 0.5

    ' Get point on surface at param .5,.5
    Dim points() As Double = New Double() {}
    Call oFace.Evaluator.GetPointAtParam(params, points)

    ' Create point object
    Dim oPoint As point
    oPoint = ThisApplication.TransientGeometry.CreatePoint(points(0), points(1), points(2))

    ' Get normal at this point
    Dim normals() As Double = New Double() {}
    Call oFace.Evaluator.GetNormal(params, normals)

    ' Create normal vector object
    Dim oNormal As Vector
    oNormal = ThisApplication.TransientGeometry.CreateVector(normals(0), normals(1), normals(2))

    ' Scale vector by radius of the cylinder
    oNormal.ScaleBy(oCylinder.Radius)

    ' Find the sampler point on the normal by adding the
    ' scaled normal vector to the point at .5,.5 param.
    Dim oSamplePoint As point
    oSamplePoint = oPoint

    oSamplePoint.TranslateBy(oNormal)

    ' Check if the sample point lies on the cylinder axis.
    ' If it does, we have a hollow face.

    ' Create a line describing the cylinder axis
    Dim oAxisLine As Line
    oAxisLine = ThisApplication.TransientGeometry.CreateLine _
        (oCylinder.BasePoint, oCylinder.AxisVector.AsVector)

    'Create a line parallel to the axis passing thru the sample point.
    Dim oSampleLine As Line
    oSampleLine = ThisApplication.TransientGeometry.CreateLine _
        (oSamplePoint, oCylinder.AxisVector.AsVector)

    If oSampleLine.IsColinearTo(oAxisLine) Then
        MsgBox("Interior face.")
    Else
        MsgBox("Exterior face.")
    End If
```

Mass Properties from Part

Description

This sample demonstrates getting mass properties from a part. This sample is specific to a part document, but the MassProperties object can also be obtained from an Assembly document and from component occurrences. To run the sample you must have a part document open that contains a solid.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub GetPartMassProps()
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the mass properties object.
    Dim oMassProps As MassProperties
    Set oMassProps = oPartDoc.ComponentDefinition.MassProperties

    ' Set the accuracy to medium.
    oMassProps.Accuracy = k_Medium

    ' Display the mass properties of the part.
    Debug.Print "Area: " & oMassProps.Area

    Debug.Print "Center of Mass: " & _
                    oMassProps.CenterOfMass.X & ", " & _
```

```
                oMassProps.CenterOfMass.Y & ", " & _
                oMassProps.CenterOfMass.Z

    Debug.Print "Mass: " & oMassProps.Mass
    Dim adPrincipalMoments(1 To 3) As Double
    Call oMassProps.PrincipalMomentsOfInertia(adPrincipalMoments(1), _
                                              adPrincipalMoments(2), _
                                              adPrincipalMoments(3))
    Debug.Print "Principal Moments of Inertia: " & _
                adPrincipalMoments(1) & ", " & _
                adPrincipalMoments(2) & ", " & _
                adPrincipalMoments(3)

    Dim adRadiusOfGyration(1 To 3) As Double
    Call oMassProps.RadiusOfGyration(adRadiusOfGyration(1), _
                                     adRadiusOfGyration(2), _
                                     adRadiusOfGyration(3))
    Debug.Print "Radius of Gyration: " & _
                adRadiusOfGyration(1) & ", " & _
                adRadiusOfGyration(2) & ", " & _
                adRadiusOfGyration(3)

    Debug.Print "Volume: " & oMassProps.Volume

    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oMassProps.XYZMomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Debug.Print "Moments: "
    Debug.Print "   Ixx: " & Ixx
    Debug.Print "   Iyy: " & Iyy
    Debug.Print "   Izz: " & Izz
    Debug.Print "   Ixy: " & Ixy
    Debug.Print "   Iyz: " & Iyz
    Debug.Print "   Ixz: " & Ixz
End Sub
```

Copy Code

```
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the mass properties object.
    Dim oMassProps As MassProperties
    oMassProps = oPartDoc.ComponentDefinition.MassProperties

    ' Set the accuracy to medium.
    oMassProps.Accuracy = k_Medium

    ' Display the mass properties of the part.
    Logger.Info("Area: " & oMassProps.Area)


    Logger.Info("Center of Mass: " & _
                oMassProps.CenterOfMass.X & ", " & _
                oMassProps.CenterOfMass.Y & ", " & _
                oMassProps.CenterOfMass.Z)

    Logger.Info("Mass: " & oMassProps.Mass)

    Dim adPrincipalMoments(0 To 2) As Double
    Call oMassProps.PrincipalMomentsOfInertia(adPrincipalMoments(0), _
                                              adPrincipalMoments(1), _
                                              adPrincipalMoments(2))
    Logger.Info("Principal Moments of Inertia: " & _
                adPrincipalMoments(0) & ", " & _
                adPrincipalMoments(1) & ", " & _
                adPrincipalMoments(2))

    Dim adRadiusOfGyration(0 To 2) As Double
    Call oMassProps.RadiusOfGyration(adRadiusOfGyration(0), _
                                     adRadiusOfGyration(1), _
                                     adRadiusOfGyration(2))
    Logger.Info("Radius of Gyration: " & _
                adRadiusOfGyration(0) & ", " & _
                adRadiusOfGyration(1) & ", " & _
                adRadiusOfGyration(2))

    Logger.Info("Volume: " & oMassProps.Volume)


    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oMassProps.XYZMomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Logger.Info("Moments: ")

    Logger.Info("   Ixx: " & Ixx)

    Logger.Info("   Iyy: " & Iyy)

    Logger.Info("   Izz: " & Izz)

    Logger.Info("   Ixy: " & Ixy)

    Logger.Info("   Iyz: " & Iyz)

    Logger.Info("   Ixz: " & Ixz)
```

Computing mass properties without dirtying document

Description

This sample demonstrates getting mass properties from a part without dirtying the document (i.e. without caching the computed results in the document). This sample is specific to a part document, but the MassProperties object can also be obtained from an Assembly document and from component occurrences.

Code Samples

- VBA
- iLogic

To run the sample you must have a part document open that contains a solid.

Copy Code

```
Public Sub GetPartMassPropsWithoutDirtying()
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the mass properties object.
    Dim oMassProps As MassProperties
    Set oMassProps = oPartDoc.ComponentDefinition.MassProperties

    'Check if mass property results are already available
    'at a high accuracy level or better. If so, simply
    'print out the results, else, set a flag to not cache
    'the results in the document.
    If oMassProps.AvailableAccuracy <> k_High And _
      oMassProps.AvailableAccuracy <> k_VeryHigh Then
      ' Set the accuracy to high.
      oMassProps.Accuracy = k_High

      'Set CacheResultsOnCompute property to False
      'so that results are not saved with the document
      'and hence the document is not 'dirtied'.
      oMassProps.CacheResultsOnCompute = False
    End If

    ' Display the mass properties of the part.
    Debug.Print "Area: " & oMassProps.Area

    Debug.Print "Center of Mass: " & _
    oMassProps.CenterOfMass.X & ", " & _
    oMassProps.CenterOfMass.Y & ", " & _
    oMassProps.CenterOfMass.Z

    Debug.Print "Mass: " & oMassProps.Mass
    Dim adPrincipalMoments(1 To 3) As Double
    Call oMassProps.PrincipalMomentsOfInertia(adPrincipalMoments(1), _
    adPrincipalMoments(2), _
    adPrincipalMoments(3))
    Debug.Print "Principal Moments of Inertia: " & _
    adPrincipalMoments(1) & ", " & _
    adPrincipalMoments(2) & ", " & _
    adPrincipalMoments(3)

    Dim adRadiusOfGyration(1 To 3) As Double
    Call oMassProps.RadiusOfGyration(adRadiusOfGyration(1), _
    adRadiusOfGyration(2), _
    adRadiusOfGyration(3))
    Debug.Print "Radius of Gyration: " & _
    adRadiusOfGyration(1) & ", " & _
    adRadiusOfGyration(2) & ", " & _
    adRadiusOfGyration(3)

    Debug.Print "Volume: " & oMassProps.Volume

    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oMassProps.XYZMomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Debug.Print "Moments: "
    Debug.Print " Ixx: " & Ixx
    Debug.Print " Iyy: " & Iyy
    Debug.Print " Izz: " & Izz
    Debug.Print " Ixy: " & Ixy
    Debug.Print " Iyz: " & Iyz
    Debug.Print " Ixz: " & Ixz
End Sub
```

To run the sample you must have a part document open that contains a solid.

Copy Code

```
    ' Set a reference to the part document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the mass properties object.
    Dim oMassProps As MassProperties
    oMassProps = oPartDoc.ComponentDefinition.MassProperties
```

```
    'Check if mass property results are already available
    'at a high accuracy level or better. If so, simply
    'print out the results, else, set a flag to not cache
    'the results in the document.
    If oMassProps.AvailableAccuracy <> k_High And _
      oMassProps.AvailableAccuracy <> k_VeryHigh Then
      ' Set the accuracy to high.
      oMassProps.Accuracy = k_High

      'Set CacheResultsOnCompute property to False
      'so that results are not saved with the document
      'and hence the document is not 'dirtied'.
      oMassProps.CacheResultsOnCompute = False
    End If

    ' Display the mass properties of the part.
    Logger.Info("Area: " & oMassProps.Area)


    Logger.Info("Center of Mass: " & _
    oMassProps.CenterOfMass.X & ", " & _
    oMassProps.CenterOfMass.Y & ", " & _
    oMassProps.CenterOfMass.Z)

    Logger.Info("Mass: " & oMassProps.Mass)

    Dim adPrincipalMoments(0 To 2) As Double
    Call oMassProps.PrincipalMomentsOfInertia(adPrincipalMoments(0), _
    adPrincipalMoments(1), _
    adPrincipalMoments(2))
    Logger.Info("Principal Moments of Inertia: " & _
    adPrincipalMoments(0) & ", " & _
    adPrincipalMoments(1) & ", " & _
    adPrincipalMoments(2))

    Dim adRadiusOfGyration(0 To 2) As Double
    Call oMassProps.RadiusOfGyration(adRadiusOfGyration(0), _
    adRadiusOfGyration(1), _
    adRadiusOfGyration(2))
    Logger.Info("Radius of Gyration: " & _
    adRadiusOfGyration(0) & ", " & _
    adRadiusOfGyration(1) & ", " & _
    adRadiusOfGyration(2))

    Logger.Info("Volume: " & oMassProps.Volume)


    Dim Ixx As Double
    Dim Iyy As Double
    Dim Izz As Double
    Dim Ixy As Double
    Dim Iyz As Double
    Dim Ixz As Double
    Call oMassProps.XYZMomentsOfInertia(Ixx, Iyy, Izz, Ixy, Iyz, Ixz)
    Logger.Info("Moments: ")

    Logger.Info(" Ixx: " & Ixx)

    Logger.Info(" Iyy: " & Iyy)

    Logger.Info(" Izz: " & Izz)

    Logger.Info(" Ixy: " & Ixy)

    Logger.Info(" Iyz: " & Iyz)

    Logger.Info(" Ixz: " & Ixz)
```

Work point at mass center

Description

This sample demonstrates creating a fixed work point and edit its position. It does this by computing the the center of mass of the part and creating a work point at that position. Subsequent runs of the sample reposition the work point at the new center of mass.

Code Samples

- C#
- VBA
- iLogic

The first line of this sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void WorkPointAtMassCenter()
{
    Application oApp = ThisApplication;

    // Check to make sure a part document is active.
    if (oApp.ActiveDocumentType != DocumentTypeEnum.kPartDocumentObject)
    {
        System.Windows.Forms.MessageBox.Show("A part document must be active.", "Part not active");
        return;
    }

    // Set a reference to the active document.
```

```
    PartDocument oDoc = (PartDocument)oApp.ActiveDocument;

    // Get the Center of Mass.
    Point oCenterOfMass = oDoc.ComponentDefinition.MassProperties.CenterOfMass;

    // Check to see if a work point for center of mass already exists.
    // This uses the name of the work feature to identify it.

    WorkPoint oWorkPoint = null;
    try
    {
        oWorkPoint = oDoc.ComponentDefinition.WorkPoints["Center Of Mass"];

        FixedWorkPointDef oFixedDef = (FixedWorkPointDef)oWorkPoint.Definition;
        oFixedDef.Point = oCenterOfMass;
        oDoc.Update();
    }
    catch
    {
        // Create a new workpoint at the location of the center of mass.
        oWorkPoint = oDoc.ComponentDefinition.WorkPoints.AddFixed(oCenterOfMass, false);

        // Rename the work point.
        oWorkPoint.Name = "Center Of Mass";
    }
}
```

Copy Code

```
Public Sub WorkPointAtMassCenter()
    ' Check to make sure a part document is active.
    If ThisApplication.ActiveDocumentType <> kPartDocumentObject Then
        MsgBox "A part document must be active."
        Exit Sub
    End If

    ' Set a reference to the active document.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Get the Center of Mass.
    Dim oCenterOfMass As Point
    Set oCenterOfMass = oDoc.ComponentDefinition.MassProperties.CenterOfMass

    ' Check to see if a work point for center of mass already exists.
    ' This uses the name of the work feature to identify it.
    On Error Resume Next
    Dim oWorkPoint As WorkPoint
    Set oWorkPoint = oDoc.ComponentDefinition.WorkPoints.Item("Center Of Mass")
    If Err.Number = 0 Then
        Dim oFixedDef As FixedWorkPointDef
        Set oFixedDef = oWorkPoint.Definition

        oFixedDef.Point = oCenterOfMass
        oDoc.Update
    Else
        ' Create a new workpoint at the location of the center of mass.
        Set oWorkPoint = oDoc.ComponentDefinition.WorkPoints.AddFixed(oCenterOfMass)

        ' Rename the work point.
        oWorkPoint.Name = "Center Of Mass"
    End If
End Sub
```

Copy Code

```
    ' Check to make sure a part document is active.
    If ThisApplication.ActiveDocumentType <> kPartDocumentObject Then
        MsgBox("A part document must be active.")
        Exit Sub
    End If

    ' Set a reference to the active document.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    ' Get the Center of Mass.
    Dim oCenterOfMass As Point
    oCenterOfMass = oDoc.ComponentDefinition.MassProperties.CenterOfMass

    ' Check to see if a work point for center of mass already exists.
    ' This uses the name of the work feature to identify it.
    On Error Resume Next
    Dim oWorkPoint As WorkPoint
    oWorkPoint = oDoc.ComponentDefinition.WorkPoints.Item("Center Of Mass")
    If Err.Number = 0 Then
        Dim oFixedDef As FixedWorkPointDef
        oFixedDef = oWorkPoint.Definition

        oFixedDef.Point = oCenterOfMass
        oDoc.Update
    Else
        ' Create a new workpoint at the location of the center of mass.
        oWorkPoint = oDoc.ComponentDefinition.WorkPoints.AddFixed(oCenterOfMass)

        ' Rename the work point.
        oWorkPoint.Name = "Center Of Mass"
    End If
```

Modify Multiple Model States Sample

Description

This sample demonstrates how to set multiple but not all model states into edit mode.

Code Samples

- VBA
- iLogic

This sample demonstrates how to set multiple but not all model states into edit mode.

Copy Code

```
Sub ModifyMultipleModelStatesSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    Dim oModelStates As ModelStates
    Set oModelStates = oCompDef.ModelStates

    ' New several model state objects
    Dim i As Long
    For i = 1 To 4
        oModelStates.Add
    Next

    Dim oCol As ObjectCollection
    Set oCol = ThisApplication.TransientObjects.CreateObjectCollection

    ' Say we would like to edit the model state from the second one to the fourth one.
    ' we should make sure the active model state is added to the edit mode
    Dim oMS As ModelState
    Set oMS = oModelStates.Item(2)
    oMS.Activate

    For i = 2 To 4
        oCol.Add oModelStates.Item(i)
    Next

    oModelStates.ModelStatesInEdit = oCol

    ' When multiple model states but not all in edit mode
    ' the MemberEditScope will be set to kEditMultipleMembers.
    Debug.Print oModelStates.MemberEditScope = kEditMultipleMembers
End Sub
```

This sample demonstrates how to set multiple but not all model states into edit mode.

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    Dim oModelStates As ModelStates
    oModelStates = oCompDef.ModelStates

    ' New several model state objects
    Dim i As Long
    For i = 1 To 4
        oModelStates.Add
    Next

    Dim oCol As ObjectCollection
    oCol = ThisApplication.TransientObjects.CreateObjectCollection

    ' Say we would like to edit the model state from the second one to the fourth one.
    ' we should make sure the active model state is added to the edit mode
    Dim oMS As ModelState
    oMS = oModelStates.Item(2)
    oMS.Activate

    For i = 2 To 4
        oCol.Add(oModelStates.Item(i))
    Next

    oModelStates.ModelStatesInEdit = oCol

    ' When multiple model states but not all in edit mode
    ' the MemberEditScope will be set to kEditMultipleMembers.
    Logger.Info(oModelStates.MemberEditScope = kEditMultipleMembers)
```

Model Parameters

Description

This sample demonstrates the methods and properties supported by the Parameters object for model parameters.

Code Samples

- VBA
- iLogic

To run the sample you must have an part document open that contains at least one parameter, one of which is named d0.

Copy Code

```
Public Sub ModelParameters()
    ' Obtain the active document, this assumes
    ' that a part document is active in Inventor.
    Dim oPartDoc As Inventor.PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Obtain the Parameters collection
    Dim oParams As Parameters
    Set oParams = oPartDoc.ComponentDefinition.Parameters

    ' Iterate through the Parameters collection to obtain
    ' information about the Parameters
    Dim iNumParams As Long
    Debug.Print "ALL PARAMETERS"
    For iNumParams = 1 To oParams.Count
        'Display the Name
        Debug.Print " Name: " & oParams.Item(iNumParams).Name

        'Display the Parameter Type
        Select Case oParams.Item(iNumParams).Type
            Case kModelParameterObject
                Debug.Print "  Type: " & "Model Parameter"
            Case kTableParameterObject
                Debug.Print "  Type: " & "Table Parameter"
            Case kUserParameterObject
                Debug.Print "  Type: " & "User Parameter"
        End Select

        'Display the Value
        Debug.Print "  Value: " & oParams.Item(iNumParams).Value

        'Display the Health Status
        Select Case oParams.Item(iNumParams).HealthStatus
            Case kDeletedHealth
                Debug.Print "  Health Status: " & "Deleted"
            Case kDriverLostHealth
                Debug.Print "  Health Status: " & "Driver Lost"
            Case kInErrorHealth
                Debug.Print "  Health Status: " & "In Error"
            Case kOutOfDateHealth
                Debug.Print "  Health Status: " & "Out of Date"
            Case kUnknownHealth
                Debug.Print "  Health Status: " & "Unknown"
            Case kUpToDateHealth
                Debug.Print "  Health Status: " & "Up to Date"
        End Select
    Next iNumParams

    ' Obtain the Model Parameters collection
    Dim oModelParams As ModelParameters
    Set oModelParams = oParams.ModelParameters

    ' Iterate through the Model Parameters collection
    Dim iNumModelParams As Long
    Debug.Print "MODEL PARAMETER VALUES"
    For iNumModelParams = 1 To oModelParams.Count
        ' Display the Name
        Debug.Print " Name:" & oModelParams.Item(iNumModelParams).Name

        ' Display the Value
        Debug.Print "  Value: " & oModelParams.Item(iNumModelParams).Value

        ' Display the units
        Debug.Print "  Units: " & oModelParams.Item(iNumModelParams).Units

        ' Change the Model Parameter values
        oModelParams.Item(iNumModelParams).Value = oModelParams.Item(iNumModelParams).Value * 2
    Next iNumModelParams

    ' Accessing a particular parameter if you know its name, the user and reference parameters can also be accessed in a similar way
    oModelParams.Item("d0").Name = "NewParam"

    ' Change the value of the newly named parameter "param1"
    oModelParams.Item("NewParam").Value = 25

    ' Update the model.
    oPartDoc.Update
End Sub
```

To run the sample you must have an part document open that contains at least one parameter, one of which is named d0.

Copy Code

```
    ' Obtain the active document, this assumes
    ' that a part document is active in Inventor.
    Dim oPartDoc As Inventor.PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Obtain the Parameters collection
    Dim oParams As Parameters
    oParams = oPartDoc.ComponentDefinition.Parameters

    ' Iterate through the Parameters collection to obtain
    ' information about the Parameters
    Dim iNumParams As Long
    Logger.Info("ALL PARAMETERS")
```

```
For iNumParams = 1 To oParams.Count
    'Display the Name
    Logger.Info(" Name: " & oParams.Item(iNumParams).Name)


    'Display the Parameter Type
    Select Case oParams.Item(iNumParams).Type
        Case kModelParameterObject
            Logger.Info("  Type: " & "Model Parameter")

        Case kTableParameterObject
            Logger.Info("  Type: " & "Table Parameter")

        Case kUserParameterObject
            Logger.Info("  Type: " & "User Parameter")

    End Select

    'Display the Value
    Logger.Info("  Value: " & oParams.Item(iNumParams).Value)


    'Display the Health Status
    Select Case oParams.Item(iNumParams).HealthStatus
        Case kDeletedHealth
            Logger.Info("  Health Status: " & "Deleted")

        Case kDriverLostHealth
            Logger.Info("  Health Status: " & "Driver Lost")

        Case kInErrorHealth
            Logger.Info("  Health Status: " & "In Error")

        Case kOutOfDateHealth
            Logger.Info("  Health Status: " & "Out of Date")

        Case kUnknownHealth
            Logger.Info("  Health Status: " & "Unknown")

        Case kUpToDateHealth
            Logger.Info("  Health Status: " & "Up to Date")

    End Select
Next iNumParams

' Obtain the Model Parameters collection
Dim oModelParams As ModelParameters
oModelParams = oParams.ModelParameters

' Iterate through the Model Parameters collection
Dim iNumModelParams As Long
Logger.Info("MODEL PARAMETER VALUES")

For iNumModelParams = 1 To oModelParams.Count
    ' Display the Name
    Logger.Info(" Name:" & oModelParams.Item(iNumModelParams).Name)


    ' Display the Value
    Logger.Info("  Value: " & oModelParams.Item(iNumModelParams).Value)


    ' Display the units
    Logger.Info("  Units: " & oModelParams.Item(iNumModelParams).Units)


    ' Change the Model Parameter values
    oModelParams.Item(iNumModelParams).Value = oModelParams.Item(iNumModelParams).Value * 2
Next iNumModelParams

' Accessing a particular parameter if you know its name, the user and reference parameters can also be accessed in a similar way
oModelParams.Item("d0").Name = "NewParam"

' Change the value of the newly named parameter "param1"
oModelParams.Item("NewParam").Value = 25

' Update the model.
oPartDoc.Update
```

Table Parameters

Description

This sample demonstrates how to access the Parameters object, and from it in turn the TableParameters collection that represents the collection of parameters that have been linked/embedded from an external spreadsheet.

Code Samples

- VBA
- iLogic

To run the sample, you need to have a part document open, and the spreadsheet C:\Temp\params.xls must exist. You can also edit the sample to reference another spreadsheet if you prefer. The spreadsheet should contain valid definitions for parameters. For the section of the code that demonstrates changing the reference you also need to create a spreadsheet C:\Temp\newparams.xls, or edit the sample to reference another spreadsheet.

Copy Code

```
Public Sub TableParameters()
    Dim oPartDoc As Inventor.PartDocument
    ' Obtain the active document, this assumes that
    ' a part document is active in Inventor
    Set oPartDoc = ThisApplication.ActiveDocument

    'Obtain the Parameters collection
    Dim oParams As Parameters
    Set oParams = oPartDoc.ComponentDefinition.Parameters

    ' Add a parameter table using an existing spreadsheet.
    oParams.ParameterTables.AddExcelTable "C:\Temp\params.xls", "A1", True

    ' Accessing a parameters in a linked/embedded file
    Dim oParamTableFiles As ParameterTables
    Set oParamTableFiles = oParams.ParameterTables

    ' Traverse through the collection of linked files
    Dim oParamTableFile As ParameterTable
    For Each oParamTableFile In oParamTableFiles
        ' Change the linked file to another file
        If LCase(oParamTableFile.FileName) = "C:\temp\params.xls" Then
            oParamTableFile.FileName = "C:\Temp\newparams.xls"
        End If

        ' Get the Parameters collection from the file
        Dim oTableParams As TableParameters
        Set oTableParams = oParamTableFile.TableParameters

        ' Traverse through the table parameters collection and display them
        Dim iNumTableParams As Long
        Debug.Print "TABLE PARAMETER VALUES"
        For iNumTableParams = 1 To oTableParams.Count
            ' Display the name
            Debug.Print " Name: " & oTableParams.Item(iNumTableParams).Name

            ' Display the expression
            Debug.Print " Expression: " & oTableParams.Item(iNumTableParams).Expression

            ' Display the value.  This will be in database units.
            Debug.Print " Value: " & oTableParams.Item(iNumTableParams).Value
        Next iNumTableParams
    Next
End Sub
```

To run the sample, you need to have a part document open, and the spreadsheet C:\Temp\params.xls must exist. You can also edit the sample to reference another spreadsheet if you prefer. The spreadsheet should contain valid definitions for parameters. For the section of the code that demonstrates changing the reference you also need to create a spreadsheet C:\Temp\newparams.xls, or edit the sample to reference another spreadsheet.

Copy Code

```
    Dim oPartDoc As Inventor.PartDocument
    ' Obtain the active document, this assumes that
    ' a part document is active in Inventor
    oPartDoc = ThisApplication.ActiveDocument

    'Obtain the Parameters collection
    Dim oParams As Parameters
    oParams = oPartDoc.ComponentDefinition.Parameters

    ' Add a parameter table using an existing spreadsheet.
    oParams.ParameterTables.AddExcelTable("C:\Temp\params.xls", "A1", True)

    ' Accessing a parameters in a linked/embedded file
    Dim oParamTableFiles As ParameterTables
    oParamTableFiles = oParams.ParameterTables

    ' Traverse through the collection of linked files
    Dim oParamTableFile As ParameterTable
    For Each oParamTableFile In oParamTableFiles
        ' Change the linked file to another file
        If LCase(oParamTableFile.FileName) = "C:\temp\params.xls" Then
            oParamTableFile.FileName = "C:\Temp\newparams.xls"
        End If

        ' Get the Parameters collection from the file
        Dim oTableParams As TableParameters
        oTableParams = oParamTableFile.TableParameters

        ' Traverse through the table parameters collection and display them
        Dim iNumTableParams As Long
        Logger.Info("TABLE PARAMETER VALUES")

        For iNumTableParams = 1 To oTableParams.Count
            ' Display the name
            Logger.Info(" Name: " & oTableParams.Item(iNumTableParams).Name)


            ' Display the expression
            Logger.Info(" Expression: " & oTableParams.Item(iNumTableParams).Expression)


            ' Display the value.  This will be in database units.
            Logger.Info(" Value: " & oTableParams.Item(iNumTableParams).Value)

        Next iNumTableParams
    Next
```

Transient surface body creation

Description

The following sample demonstrates the creation of a transient surface body consisting of a single rectangular face. The body is created in transient space and then copied over to a part document as a base feature.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateRectangleFace()
    Dim oTransBRep As TransientBRep
    Set oTransBRep = ThisApplication.TransientBRep

    Dim oSurfaceBodyDef As SurfaceBodyDefinition
    Set oSurfaceBodyDef = oTransBRep.CreateSurfaceBodyDefinition

    ' Create a lump.
    Dim oLumpDef As LumpDefinition
    Set oLumpDef = oSurfaceBodyDef.LumpDefinitions.Add

    ' Create a shell.
    Dim oShell As FaceShellDefinition
    Set oShell = oLumpDef.FaceShellDefinitions.Add

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Create transient points representing the four corners.
    Dim oPoint1 As Point
    Set oPoint1 = oTG.CreatePoint(-1, -1, 1)
    Dim oPoint2 As Point
    Set oPoint2 = oTG.CreatePoint(1, -1, 1)
    Dim oPoint3 As Point
    Set oPoint3 = oTG.CreatePoint(1, 1, 1)
    Dim oPoint4 As Point
    Set oPoint4 = oTG.CreatePoint(-1, 1, 1)

    ' Create the Vertices.
    Dim oVertex1 As VertexDefinition
    Set oVertex1 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint1)
    Dim oVertex2 As VertexDefinition
    Set oVertex2 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint2)
    Dim oVertex3 As VertexDefinition
    Set oVertex3 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint3)
    Dim oVertex4 As VertexDefinition
    Set oVertex4 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint4)

    ' Create each of the edges, as defined by a line segment.
    Dim oLineSeg As LineSegment
    Set oLineSeg = oTG.CreateLineSegment(oPoint1, oPoint2)

    Dim oEdgeDef1 As EdgeDefinition
    Set oEdgeDef1 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex1, oVertex2, oLineSeg)

    Set oLineSeg = oTG.CreateLineSegment(oPoint2, oPoint3)

    Dim oEdgeDef2 As EdgeDefinition
    Set oEdgeDef2 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex3, oLineSeg)

    Set oLineSeg = oTG.CreateLineSegment(oPoint3, oPoint4)

    Dim oEdgeDef3 As EdgeDefinition
    Set oEdgeDef3 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex3, oVertex4, oLineSeg)

    Set oLineSeg = oTG.CreateLineSegment(oPoint4, oPoint1)

    Dim oEdgeDef4 As EdgeDefinition
    Set oEdgeDef4 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex1, oLineSeg)

    ' Create the face as a planar face.
    Dim oFaceDef As FaceDefinition
    Set oFaceDef = oShell.FaceDefinitions.Add(oTG.CreatePlane( _
                                              oTG.CreatePoint(0, 0, 1), _
                                              oTG.CreateVector(0, 0, 1)), False)

    ' Create the loop.
    Dim oEdgeLoop As EdgeLoopDefinition
    Set oEdgeLoop = oFaceDef.EdgeLoopDefinitions.Add

    ' Define each of the edge uses of the loop using the previously defined edges.
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef1, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef2, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef3, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef4, False)

    ' Create a transient body.
    Dim oErrors As NameValueMap
    Dim oNewBody As SurfaceBody
    Set oNewBody = oSurfaceBodyDef.CreateTransientSurfaceBody(oErrors)

    ' Create a non-parametric base feature based on the transient body.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oDef As PartComponentDefinition
    Set oDef = oDoc.ComponentDefinition
```

```
    Dim oFeatureDef As NonParametricBaseFeatureDefinition
    Set oFeatureDef = oDef.Features.NonParametricBaseFeatures.CreateDefinition

    Dim oCollection As ObjectCollection
    Set oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    oCollection.Add oNewBody

    oFeatureDef.BRepEntities = oCollection
    oFeatureDef.OutputType = kSurfaceOutputType

    Dim oBaseFeature As NonParametricBaseFeature
    Set oBaseFeature = oDef.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef)
End Sub
```

Copy Code

```
    Dim oTransBRep As TransientBRep
    oTransBRep = ThisApplication.TransientBRep

    Dim oSurfaceBodyDef As SurfaceBodyDefinition
    oSurfaceBodyDef = oTransBRep.CreateSurfaceBodyDefinition

    ' Create a lump.
    Dim oLumpDef As LumpDefinition
    oLumpDef = oSurfaceBodyDef.LumpDefinitions.Add

    ' Create a shell.
    Dim oShell As FaceShellDefinition
    oShell = oLumpDef.FaceShellDefinitions.Add

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Create transient points representing the four corners.
    Dim oPoint1 As Point
    oPoint1 = oTG.CreatePoint(-1, -1, 1)
    Dim oPoint2 As Point
    oPoint2 = oTG.CreatePoint(1, -1, 1)
    Dim oPoint3 As Point
    oPoint3 = oTG.CreatePoint(1, 1, 1)
    Dim oPoint4 As Point
    oPoint4 = oTG.CreatePoint(-1, 1, 1)

    ' Create the Vertices.
    Dim oVertex1 As VertexDefinition
    oVertex1 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint1)
    Dim oVertex2 As VertexDefinition
    oVertex2 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint2)
    Dim oVertex3 As VertexDefinition
    oVertex3 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint3)
    Dim oVertex4 As VertexDefinition
    oVertex4 = oSurfaceBodyDef.VertexDefinitions.Add(oPoint4)

    ' Create each of the edges, as defined by a line segment.
    Dim oLineSeg As LineSegment
    oLineSeg = oTG.CreateLineSegment(oPoint1, oPoint2)

    Dim oEdgeDef1 As EdgeDefinition
    oEdgeDef1 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex1, oVertex2, oLineSeg)

    oLineSeg = oTG.CreateLineSegment(oPoint2, oPoint3)

    Dim oEdgeDef2 As EdgeDefinition
    oEdgeDef2 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex2, oVertex3, oLineSeg)

    oLineSeg = oTG.CreateLineSegment(oPoint3, oPoint4)

    Dim oEdgeDef3 As EdgeDefinition
    oEdgeDef3 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex3, oVertex4, oLineSeg)

    oLineSeg = oTG.CreateLineSegment(oPoint4, oPoint1)

    Dim oEdgeDef4 As EdgeDefinition
    oEdgeDef4 = oSurfaceBodyDef.EdgeDefinitions.Add(oVertex4, oVertex1, oLineSeg)

    ' Create the face as a planar face.
    Dim oFaceDef As FaceDefinition
    oFaceDef = oShell.FaceDefinitions.Add(oTG.CreatePlane( _
                                          oTG.CreatePoint(0, 0, 1), _
                                          oTG.CreateVector(0, 0, 1)), False)

    ' Create the loop.
    Dim oEdgeLoop As EdgeLoopDefinition
    oEdgeLoop = oFaceDef.EdgeLoopDefinitions.Add

    ' Define each of the edge uses of the loop using the previously defined edges.
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef1, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef2, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef3, False)
    Call oEdgeLoop.EdgeUseDefinitions.Add(oEdgeDef4, False)

    ' Create a transient body.
    Dim oErrors As NameValueMap
    Dim oNewBody As SurfaceBody
    oNewBody = oSurfaceBodyDef.CreateTransientSurfaceBody(oErrors)

    ' Create a non-parametric base feature based on the transient body.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oDef As PartComponentDefinition
    oDef = oDoc.ComponentDefinition

    Dim oFeatureDef As NonParametricBaseFeatureDefinition
    oFeatureDef = oDef.Features.NonParametricBaseFeatures.CreateDefinition
```

```
    Dim oCollection As ObjectCollection
    oCollection = ThisApplication.TransientObjects.CreateObjectCollection

    oCollection.Add(oNewBody)

    oFeatureDef.BRepEntities = oCollection
    oFeatureDef.OutputType = kSurfaceOutputType

    Dim oBaseFeature As NonParametricBaseFeature
    oBaseFeature = oDef.Features.NonParametricBaseFeatures.AddByDefinition(oFeatureDef)
```

Create primitive BRep

Description

This sample demonstrates the creation of primitive (solid) BRep.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateBRep()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Set a reference to the TransientBRep object.
    Dim oTransientBRep As TransientBRep
    Set oTransientBRep = ThisApplication.TransientBRep

    ' Create a range box that will define the extents of a block.
    Dim oBox As Box
    Set oBox = ThisApplication.TransientGeometry.CreateBox

    ' Expand in all directions by 1 cm.
    oBox.Expand 1

    ' Create the block.
    Dim oBody As SurfaceBody
    Set oBody = oTransientBRep.CreateSolidBlock(oBox)

    ' Create bottom and top points for a cylinder.
    Dim oBottomPt As Point
    Set oBottomPt = ThisApplication.TransientGeometry.CreatePoint(0, 1, 0)

    Dim oTopPt As Point
    Set oTopPt = ThisApplication.TransientGeometry.CreatePoint(0, 3, 0)

    ' Create the cylinder body.
    Dim oCylinder As SurfaceBody
    Set oCylinder = oTransientBRep.CreateSolidCylinderCone(oBottomPt, oTopPt, 0.5, 0.5, 0.5)

    ' Boolean the bodies; "oBody" will return the result
    Call oTransientBRep.DoBoolean(oBody, oCylinder, kBooleanTypeUnion)

    ' Create a base feature with the result body.
    Dim oBaseFeature As NonParametricBaseFeature
    Set oBaseFeature = oCompDef.Features.NonParametricBaseFeatures.Add(oBody)
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Set a reference to the TransientBRep object.
    Dim oTransientBRep As TransientBRep
    oTransientBRep = ThisApplication.TransientBRep

    ' Create a range box that will define the extents of a block.
    Dim oBox As Box
    oBox = ThisApplication.TransientGeometry.CreateBox

    ' Expand in all directions by 1 cm.
    oBox.Expand(1)

    ' Create the block.
    Dim oBody As SurfaceBody
    oBody = oTransientBRep.CreateSolidBlock(oBox)

    ' Create bottom and top points for a cylinder.
    Dim oBottomPt As Point
```

```
    oBottomPt = ThisApplication.TransientGeometry.CreatePoint(0, 1, 0)

    Dim oTopPt As Point
    oTopPt = ThisApplication.TransientGeometry.CreatePoint(0, 3, 0)

    ' Create the cylinder body.
    Dim oCylinder As SurfaceBody
    oCylinder = oTransientBRep.CreateSolidCylinderCone(oBottomPt, oTopPt, 0.5, 0.5, 0.5)

    ' Boolean the bodies; "oBody" will return the result
    Call oTransientBRep.DoBoolean(oBody, oCylinder, kBooleanTypeUnion)

    ' Create a base feature with the result body.
    Dim oBaseFeature As NonParametricBaseFeature
    oBaseFeature = oCompDef.Features.NonParametricBaseFeatures.Add(oBody)
```

Edit width extent of an existing flange feature

Description

This sample demonstrates editing the width extent of an existing flange feature. This expects an existing sheet metal document that contains a flange feature that contains for physical flanges. It changes the type of width extent for each of the physical flanges. The result from the FlangeWidthsCreation sample can be used as the document to run this macro in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub EditFlangeWidths()
    ' Create the active sheet metal document.
    Dim sheetMetalDoc As PartDocument
    Set sheetMetalDoc = ThisApplication.ActiveDocument

    ' Get the sheet metal component definition.
    Dim smCompDef As SheetMetalComponentDefinition
    Set smCompDef = sheetMetalDoc.ComponentDefinition

    Dim smFeatures As SheetMetalFeatures
    Set smFeatures = smCompDef.Features

    ' Get the flange feature.
    Dim flange As FlangeFeature
    Set flange = smFeatures.FlangeFeatures.Item(1)

    ' Position the end of part marker to just before this feature so the
    ' edges are in their original state.
    Call flange.SetEndOfPart(True)

    Dim flangeDef As FlangeDefinition
    Set flangeDef = flange.Definition

    ' Iterate over the four edges and edit the width extent for each one.
    Dim i As Integer
    For i = 1 To 4
        Dim currentEdge As Edge
        Set currentEdge = flangeDef.Edges.Item(i)
        Select Case i
            Case 1
                ' Edit the width extent to be offset as measured from the vertices of the edge.
                Call flangeDef.SetOffsetWidthExtent(currentEdge, currentEdge.StartVertex, 2, currentEdge.StopVertex, 5)
            Case 2
                Call flangeDef.SetEdgeWidthExtent(currentEdge)
            Case 3
                ' Edit the width extent to be an offset width extent.
                Call flangeDef.SetWidthOffsetWidthExtent(currentEdge, 6, 2, currentEdge.StartVertex, True)
            Case 4
                ' Edit the width extent to be centered.
                Call flangeDef.SetCenteredWidthExtent(currentEdge, 10)
        End Select
    Next

    ' Set the stop node back to the bottom of the browser.
    Call smCompDef.SetEndOfPartToTopOrBottom(False)
End Sub
```

Copy Code

```
    ' Create the active sheet metal document.
    Dim sheetMetalDoc As PartDocument
    sheetMetalDoc = ThisApplication.ActiveDocument

    ' Get the sheet metal component definition.
    Dim smCompDef As SheetMetalComponentDefinition
    smCompDef = sheetMetalDoc.ComponentDefinition

    Dim smFeatures As SheetMetalFeatures
    smFeatures = smCompDef.Features

    ' Get the flange feature.
    Dim flange As FlangeFeature
    flange = smFeatures.FlangeFeatures.Item(1)

    ' Position the end of part marker to just before this feature so the
    ' edges are in their original state.
```

```
    Call flange.SetEndOfPart(True)

    Dim flangeDef As FlangeDefinition
    flangeDef = flange.Definition

    ' Iterate over the four edges and edit the width extent for each one.
    Dim i As Integer
    For i = 1 To 4
        Dim currentEdge As Edge
        currentEdge = flangeDef.Edges.Item(i)
        Select Case i
            Case 1
                ' Edit the width extent to be offset as measured from the vertices of the edge.
                Call flangeDef.SetOffsetWidthExtent(currentEdge, currentEdge.StartVertex, 2, currentEdge.StopVertex, 5)
            Case 2
                Call flangeDef.SetEdgeWidthExtent(currentEdge)
            Case 3
                ' Edit the width extent to be an offset width extent.
                Call flangeDef.SetWidthOffsetWidthExtent(currentEdge, 6, 2, currentEdge.StartVertex, True)
            Case 4
                ' Edit the width extent to be centered.
                Call flangeDef.SetCenteredWidthExtent(currentEdge, 10)
        End Select
    Next

    ' Set the stop node back to the bottom of the browser.
    Call smCompDef.SetEndOfPartToTopOrBottom(False)
```

Creating flange features

Description

Demonstrates creating flange features of various width extents. This creates a new document, creates a face feature and adds a flange feature on four edges.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub FlangeWidthsCreation()
    ' Create a new sheet metal document.
    Dim sheetMetalDoc As PartDocument
    Set sheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                        ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}")

    ' Get the sheet metal component definition.
    Dim smCompDef As SheetMetalComponentDefinition
    Set smCompDef = sheetMetalDoc.ComponentDefinition

    ' Create a sketch on the x-y plane.
    Dim sketch As PlanarSketch
    Set sketch = smCompDef.Sketches.Add(smCompDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    Set tg = ThisApplication.TransientGeometry

    ' Draw a rectangle.
    Call sketch.SketchLines.AddAsTwoPointRectangle(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(25, 15))

    ' Create a profile.
    Dim profile As profile
    Set profile = sketch.Profiles.AddForSolid

    Dim smFeatures As SheetMetalFeatures
    Set smFeatures = smCompDef.Features

    ' Build up a face feature definition.
    Dim faceDef As FaceFeatureDefinition
    Set faceDef = smFeatures.FaceFeatures.CreateFaceFeatureDefinition(profile)

    ' Create a face feature.
    Dim faceFeature As faceFeature
    Set faceFeature = smFeatures.FaceFeatures.Add(faceDef)

    ' Get the "top" face.
    Dim topFace As Face
    Set topFace = smCompDef.SurfaceBodies.Item(1).LocateUsingPoint(kFaceObject, tg.CreatePoint(5, 5, smCompDef.Thickness.Value))

    ' Create a collection containing the four edges.
    Dim edgeSet As EdgeCollection
    Set edgeSet = ThisApplication.TransientObjects.CreateEdgeCollection
    Dim i As Integer
    For i = 1 To 4
        Call edgeSet.Add(topFace.Edges.Item(i))
    Next

    ' Create the flange definition.
    Dim flangeDef As FlangeDefinition
    Set flangeDef = smFeatures.FlangeFeatures.CreateFlangeDefinition(edgeSet, "90", 6)

    Dim oEdgeSet As FlangeEdgeSet
    Dim oEdgeCol As EdgeCollection
    Set oEdgeCol = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Edit the definition to define a different width extent for each edge.
    For i = 1 To 4
```

```
        Dim faceEdge As Edge
        Set faceEdge = edgeSet.Item(i)
        Select Case i
            Case 1
                ' Do nothing and let it default to the edge extent.
            Case 2
                ' Edit the width extent to be centered.
                oEdgeCol.Clear
                oEdgeCol.Add faceEdge
                Set oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetCenteredWidthExtent 10
            Case 3
                ' Edit the width extent to be offset as measured from the vertices of the edge.
                oEdgeCol.Clear
                oEdgeCol.Add faceEdge
                Set oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetOffsetdWidthExtent faceEdge.StartVertex, 2, faceEdge.StopVertex, 5
            Case 4
                ' Edit the width extent to be an offset width extent.
                oEdgeCol.Clear
                oEdgeCol.Add faceEdge
                Set oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetWidthOffsetWidthExtent 6, 2, faceEdge.StartVertex, True
        End Select
    Next

    ' Create the flange.
    Call smFeatures.FlangeFeatures.Add(flangeDef)
End Sub
```

Copy Code

```
    ' Create a new sheet metal document.
    Dim sheetMetalDoc As PartDocument
    sheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                        ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}")

    ' Get the sheet metal component definition.
    Dim smCompDef As SheetMetalComponentDefinition
    smCompDef = sheetMetalDoc.ComponentDefinition

    ' Create a sketch on the x-y plane.
    Dim sketch As PlanarSketch
    sketch = smCompDef.Sketches.Add(smCompDef.WorkPlanes.Item(3))

    Dim tg As TransientGeometry
    tg = ThisApplication.TransientGeometry

    ' Draw a rectangle.
    Call sketch.SketchLines.AddAsTwoPointRectangle(tg.CreatePoint2d(0, 0), tg.CreatePoint2d(25, 15))

    ' Create a profile.
    Dim profile As Profile
    profile = sketch.Profiles.AddForSolid

    Dim smFeatures As SheetMetalFeatures
    smFeatures = smCompDef.Features

    ' Build up a face feature definition.
    Dim faceDef As FaceFeatureDefinition
    faceDef = smFeatures.FaceFeatures.CreateFaceFeatureDefinition(profile)

    ' Create a face feature.
    Dim faceFeature As FaceFeature
    faceFeature = smFeatures.FaceFeatures.Add(faceDef)

    ' Get the "top" face.
    Dim topFace As Face
    topFace = smCompDef.SurfaceBodies.Item(1).LocateUsingPoint(kFaceObject, tg.CreatePoint(5, 5, smCompDef.Thickness.Value))

    ' Create a collection containing the four edges.
    Dim edgeSet As EdgeCollection
    edgeSet = ThisApplication.TransientObjects.CreateEdgeCollection
    Dim i As Integer
    For i = 1 To 4
        Call edgeSet.Add(topFace.Edges.Item(i))
    Next

    ' Create the flange definition.
    Dim flangeDef As FlangeDefinition
    flangeDef = smFeatures.FlangeFeatures.CreateFlangeDefinition(edgeSet, "90", 6)

        Dim oEdgeSet As FlangeEdgeSet
    Dim oEdgeCol As EdgeCollection
    oEdgeCol = ThisApplication.TransientObjects.CreateEdgeCollection

    ' Edit the definition to define a different width extent for each edge.
    For i = 1 To 4
        Dim faceEdge As Edge
        faceEdge = edgeSet.Item(i)
        Select Case i
            Case 1
                ' Do nothing and let it default to the edge extent.
            Case 2
                ' Edit the width extent to be centered.
                oEdgeCol.Clear
                oEdgeCol.Add(faceEdge)
                oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetCenteredWidthExtent(10)
            Case 3
                ' Edit the width extent to be offset as measured from the vertices of the edge.
                oEdgeCol.Clear
                oEdgeCol.Add(faceEdge)
                oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetOffsetdWidthExtent(faceEdge.StartVertex, 2, faceEdge.StopVertex, 5)
            Case 4
                ' Edit the width extent to be an offset width extent.
```

```
                oEdgeCol.Clear
                oEdgeCol.Add(faceEdge)
                oEdgeSet = flangeDef.AddFlangeEdgeSet(oEdgeCol)
                oEdgeSet.SetWidthOffsetWidthExtent(6, 2, faceEdge.StartVertex, True)
            End Select
        Next

        ' Create the flange.
        Call smFeatures.FlangeFeatures.Add(flangeDef)
```

Finding Bend Extent (Tangent) Edges

Description

This sample demonstrates how to retrieve the bend extent edges (a.k.a. tangent edges) associated with the selected bend edge on a flat pattern.

Code Samples

- VBA
- iLogic

Before running the sample, activate a flat pattern and select a bend edge.

Copy Code

```
Sub BendExtentEdges()
    ' Check to make sure a flat pattern is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is FlatPattern Then
        MsgBox "A flat pattern must be active."
        Exit Sub
    End If

    ' Set a reference to the active document.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active flat pattern.
    Dim oFlatPattern As FlatPattern
    Set oFlatPattern = ThisApplication.ActiveEditObject

    ' Check to make sure a bend edge is selected.
    If Not TypeOf oPartDoc.SelectSet(1) Is Edge Then
        MsgBox "A bend edge must be selected."
        Exit Sub
    End If

    Dim oBendEdge As Edge
    Set oBendEdge = oPartDoc.SelectSet(1)

    If Not IsBendEdge(oBendEdge, oFlatPattern) Then
        MsgBox "A bend edge must be selected."
        Exit Sub
    End If

    Dim oAllTangentEdges As Edges
    Set oAllTangentEdges = oFlatPattern.GetEdgesOfType(kTangentFlatPatternEdge)

    Dim MinimumDist As Double
    MinimumDist = 0

    ' Collect up tangent edges that are parallel and closest to the bend edge
    Dim oBendTangentEdges As ObjectCollection
    Set oBendTangentEdges = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oEdge As Edge
    For Each oEdge In oAllTangentEdges
        ' Only interested in edges that are parallel to the bend edge
        If oBendEdge.Geometry.Direction.IsParallelTo(oEdge.Geometry.Direction, 0.0001) Then
            ' Eliminate any bend edges whose midpoint when projected onto
            ' the bend line is not within the length of the bend line.
            If IsWithinBendEdge(oBendEdge, oEdge) Then
                Dim dist As Double
                dist = ThisApplication.MeasureTools.GetMinimumDistance(oBendEdge, oEdge)
                If MinimumDist = 0 Then
                    oBendTangentEdges.Add oEdge
                    MinimumDist = dist
                ElseIf Abs(dist - MinimumDist) < 0.00001 Then
                    oBendTangentEdges.Add oEdge
                    MinimumDist = dist
                ElseIf dist < MinimumDist Then
                    ' Clear the collection
                    Dim oObject As Object
                    For Each oObject In oBendTangentEdges
                        oBendTangentEdges.RemoveByObject oObject
                    Next

                    oBendTangentEdges.Add oEdge
                    MinimumDist = dist
                End If
            End If
        End If
    Next

    ' Highlight tangent edges in blue.
    Dim oTangentHS As HighlightSet
    Set oTangentHS = oPartDoc.CreateHighlightSet
    oTangentHS.Color = ThisApplication.TransientObjects.CreateColor(255, 255, 0)

    Dim oBendTangentEdge As Edge
```

```
    For Each oBendTangentEdge In oBendTangentEdges
        oTangentHS.AddItem oBendTangentEdge
    Next

    MsgBox "Bend extent edges highlighted in yellow."
End Sub

Private Function IsWithinBendEdge(oBendEdge As Edge, TangentEdge As Edge) As Boolean
    Dim dPi As Double
    dPi = 4 * Atn(1)

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' Determine the lengths of the sides of the triangle defined by the end points
    ' of the bend edge and the center point of the tangent edge.
    Dim oStartPoint As Point2d
    Set oStartPoint = oTG.CreatePoint2d(oBendEdge.StartVertex.Point.x, oBendEdge.StartVertex.Point.Y)
    Dim oEndPoint As Point2d
    Set oEndPoint = oTG.CreatePoint2d(oBendEdge.StopVertex.Point.x, oBendEdge.StopVertex.Point.Y)
    Dim oTangentPoint As Point2d
    Set oTangentPoint = oTG.CreatePoint2d((TangentEdge.StartVertex.Point.x + TangentEdge.StopVertex.Point.x) / 2, (TangentEdge.StartVe
    Dim dBendLength As Double
    dBendLength = oStartPoint.DistanceTo(oEndPoint)
    Dim dStartLength As Double
    dStartLength = oStartPoint.DistanceTo(oTangentPoint)
    Dim dEndLength As Double
    dEndLength = oEndPoint.DistanceTo(oTangentPoint)

    ' Compute the angle from the start point to the tangent midpoint.
    Dim dStartAngle As Double
    dStartAngle = ArcCos((dStartLength ^ 2 + dBendLength ^ 2 - dEndLength ^ 2) / (2 * dStartLength * dBendLength))

    ' Check that it is less than 90 deg.
    If dStartAngle > dPi / 2 Then
        IsWithinBendEdge = False
        Exit Function
    End If

    ' Compute the angle from the end point to the tangent midpoint.
    Dim dEndAngle As Double
    dEndAngle = ArcCos((dEndLength ^ 2 + dBendLength ^ 2 - dStartLength ^ 2) / (2 * dEndLength * dBendLength))

    ' Check that it is less than 90 deg.
    If dEndAngle > dPi / 2 Then
        IsWithinBendEdge = False
        Exit Function
    End If

    IsWithinBendEdge = True
End Function

Private Function ArcCos(x As Double) As Double
    Dim dPi As Double
    dPi = 4 * Atn(1)

    If x = 1 Then
        ArcCos = 0
    ElseIf Abs((Abs(x) - 1)) < 0.000001 Then
        ArcCos = dPi
    Else
        ArcCos = Atn(-x / Sqr(-x * x + 1)) + 2 * (dPi / 4)
    End If
End Function

Private Function IsBendEdge(oEdge As Edge, oFlatPattern As FlatPattern) As Boolean
    ' Return False if the input edge is not a wire edge.
    Dim oWire As Wire
    Set oWire = oEdge.Wire

    If oWire Is Nothing Then
        IsBendEdge = False
        Exit Function
    End If

    Dim oAllBendEdges As EdgeCollection
    Set oAllBendEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oTempEdge As Edge

    ' Get all Bend UP edges on top face
    Dim oTopFaceBendUpEdges As Edges
    Set oTopFaceBendUpEdges = oFlatPattern.GetEdgesOfType(kBendUpFlatPatternEdge, True)
    For Each oTempEdge In oTopFaceBendUpEdges
        oAllBendEdges.Add oTempEdge
    Next

    ' Get all Bend DOWN edges on top face
    Dim oTopFaceBendDownEdges As Edges
    Set oTopFaceBendDownEdges = oFlatPattern.GetEdgesOfType(kBendDownFlatPatternEdge, True)
    For Each oTempEdge In oTopFaceBendDownEdges
        oAllBendEdges.Add oTempEdge
    Next

    ' Get all Bend UP edges on bottom face
    Dim oBottomFaceBendUpEdges As Edges
    Set oBottomFaceBendUpEdges = oFlatPattern.GetEdgesOfType(kBendUpFlatPatternEdge, False)
    For Each oTempEdge In oBottomFaceBendUpEdges
        oAllBendEdges.Add oTempEdge
    Next

    ' Get all Bend DOWN edges on bottom face
    Dim oBottomFaceBendDownEdges As Edges
    Set oBottomFaceBendDownEdges = oFlatPattern.GetEdgesOfType(kBendDownFlatPatternEdge, False)
    For Each oTempEdge In oBottomFaceBendDownEdges
        oAllBendEdges.Add oTempEdge
    Next
```

```
     ' Check if the input edge is a bend edge
    For Each oTempEdge In oAllBendEdges
        If oTempEdge Is oEdge Then
            IsBendEdge = True
            Exit Function
        End If
    Next

    IsBendEdge = False
End Function
```

Before running the sample, activate a flat pattern and select a bend edge.

Copy Code

```
Sub Main
    ' Check to make sure a flat pattern is open.
    If Not TypeOf ThisApplication.ActiveEditObject Is FlatPattern Then
        MsgBox("A flat pattern must be active.")
        Exit Sub
    End If

    ' Set a reference to the active document.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Set a reference to the active flat pattern.
    Dim oFlatPattern As FlatPattern
    oFlatPattern = ThisApplication.ActiveEditObject

    ' Check to make sure a bend edge is selected.
    If Not TypeOf oPartDoc.SelectSet(1) Is Edge Then
        MsgBox("A bend edge must be selected.")
        Exit Sub
    End If

    Dim oBendEdge As Edge
    oBendEdge = oPartDoc.SelectSet(1)

    If Not IsBendEdge(oBendEdge, oFlatPattern) Then
        MsgBox("A bend edge must be selected.")
        Exit Sub
    End If

    Dim oAllTangentEdges As Edges
    oAllTangentEdges = oFlatPattern.GetEdgesOfType(kTangentFlatPatternEdge)

    Dim MinimumDist As Double
    MinimumDist = 0

    ' Collect up tangent edges that are parallel and closest to the bend edge
    Dim oBendTangentEdges As ObjectCollection
    oBendTangentEdges = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oEdge As Edge
    For Each oEdge In oAllTangentEdges
        ' Only interested in edges that are parallel to the bend edge
                Dim oLineSeg As LineSegment
                oLineSeg = oBendEdge.Geometry

                Dim oDirection As UnitVector
                oDirection = oLineSeg.Direction

                Dim oDirection2 As UnitVector
                oDirection2 = oEdge.Geometry.Direction

        If oDirection.IsParallelTo(oDirection2, 0.0001) Then
            ' Eliminate any bend edges whose midpoint when projected onto
            ' the bend line is not within the length of the bend line.
            If IsWithinBendEdge(oBendEdge, oEdge) Then
                Dim dist As Double
                dist = ThisApplication.MeasureTools.GetMinimumDistance(oBendEdge, oEdge)
                If MinimumDist = 0 Then
                    oBendTangentEdges.Add(oEdge)
                    MinimumDist = dist
                ElseIf Abs(dist - MinimumDist) < 0.00001 Then
                    oBendTangentEdges.Add(oEdge)
                    MinimumDist = dist
                ElseIf dist < MinimumDist Then
                    ' Clear the collection
                    Dim oObject As Object
                    For Each oObject In oBendTangentEdges
                        oBendTangentEdges.RemoveByObject(oObject)
                    Next

                    oBendTangentEdges.Add(oEdge)
                    MinimumDist = dist
                End If
            End If
        End If
    Next

    ' Highlight tangent edges in blue.
    Dim oTangentHS As HighlightSet
    oTangentHS = oPartDoc.CreateHighlightSet
    oTangentHS.Color = ThisApplication.TransientObjects.CreateColor(255, 255, 0)

    Dim oBendTangentEdge As Edge
    For Each oBendTangentEdge In oBendTangentEdges
        oTangentHS.AddItem(oBendTangentEdge)
    Next

    MsgBox("Bend extent edges highlighted in yellow.")
End Sub

Private Function IsWithinBendEdge(oBendEdge As Edge, TangentEdge As Edge) As Boolean
    Dim dPi As Double
```

```
    dPi = 4 * Math.Atan(1)

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    ' Determine the lengths of the sides of the triangle defined by the end points
    ' of the bend edge and the center point of the tangent edge.
    Dim oStartPoint As Point2d
    oStartPoint = oTG.CreatePoint2d(oBendEdge.StartVertex.Point.x, oBendEdge.StartVertex.Point.Y)
    Dim oEndPoint As Point2d
    oEndPoint = oTG.CreatePoint2d(oBendEdge.StopVertex.Point.x, oBendEdge.StopVertex.Point.Y)
    Dim oTangentPoint As Point2d
    oTangentPoint = oTG.CreatePoint2d((TangentEdge.StartVertex.Point.x + TangentEdge.StopVertex.Point.x) / 2, (TangentEdge.StartVertex
    Dim dBendLength As Double
    dBendLength = oStartPoint.DistanceTo(oEndPoint)
    Dim dStartLength As Double
    dStartLength = oStartPoint.DistanceTo(oTangentPoint)
    Dim dEndLength As Double
    dEndLength = oEndPoint.DistanceTo(oTangentPoint)

    ' Compute the angle from the start point to the tangent midpoint.
    Dim dStartAngle As Double
    dStartAngle = ArcCos((dStartLength ^ 2 + dBendLength ^ 2 - dEndLength ^ 2) / (2 * dStartLength * dBendLength))

    ' Check that it is less than 90 deg.
    If dStartAngle > dPi / 2 Then
        IsWithinBendEdge = False
        Exit Function
    End If

    ' Compute the angle from the end point to the tangent midpoint.
    Dim dEndAngle As Double
    dEndAngle = ArcCos((dEndLength ^ 2 + dBendLength ^ 2 - dStartLength ^ 2) / (2 * dEndLength * dBendLength))

    ' Check that it is less than 90 deg.
    If dEndAngle > dPi / 2 Then
        IsWithinBendEdge = False
        Exit Function
    End If

    IsWithinBendEdge = True
End Function

Private Function ArcCos(x As Double) As Double
    Dim dPi As Double
    dPi = 4 * Math.Atan(1)

    If x = 1 Then
        ArcCos = 0
    ElseIf Abs((Abs(x) - 1)) < 0.000001 Then
        ArcCos = dPi
    Else
        ArcCos = Math.Atan(-x / Math.Sqrt(-x * x + 1)) + 2 * (dPi / 4)
    End If
End Function

Private Function IsBendEdge(oEdge As Edge, oFlatPattern As FlatPattern) As Boolean
    ' Return False if the input edge is not a wire edge.
    Dim oWire As Wire
    oWire = oEdge.Wire

    If oWire Is Nothing Then
        IsBendEdge = False
        Exit Function
    End If

    Dim oAllBendEdges As EdgeCollection
    oAllBendEdges = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oTempEdge As Edge

    ' Get all Bend UP edges on top face
    Dim oTopFaceBendUpEdges As Edges
    oTopFaceBendUpEdges = oFlatPattern.GetEdgesOfType(kBendUpFlatPatternEdge, True)
    For Each oTempEdge In oTopFaceBendUpEdges
        oAllBendEdges.Add(oTempEdge)
    Next

    ' Get all Bend DOWN edges on top face
    Dim oTopFaceBendDownEdges As Edges
    oTopFaceBendDownEdges = oFlatPattern.GetEdgesOfType(kBendDownFlatPatternEdge, True)
    For Each oTempEdge In oTopFaceBendDownEdges
        oAllBendEdges.Add(oTempEdge)
    Next

    ' Get all Bend UP edges on bottom face
    Dim oBottomFaceBendUpEdges As Edges
    oBottomFaceBendUpEdges = oFlatPattern.GetEdgesOfType(kBendUpFlatPatternEdge, False)
    For Each oTempEdge In oBottomFaceBendUpEdges
        oAllBendEdges.Add(oTempEdge)
    Next

    ' Get all Bend DOWN edges on bottom face
    Dim oBottomFaceBendDownEdges As Edges
    oBottomFaceBendDownEdges = oFlatPattern.GetEdgesOfType(kBendDownFlatPatternEdge, False)
    For Each oTempEdge In oBottomFaceBendDownEdges
        oAllBendEdges.Add(oTempEdge)
    Next

     ' Check if the input edge is a bend edge
    For Each oTempEdge In oAllBendEdges
        If oTempEdge Is oEdge Then
            IsBendEdge = True
            Exit Function
        End If
    Next
```

```
    IsBendEdge = False
End Function
```

Create sheet metal face and fold features

Description

This sample demonstrates the creation of sheet metal face and fold features.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub FaceAndFoldFeatureCreation()
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    Set oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    Set oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    Set oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    Set oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the new sketch.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    Set oFrontFace = oFaceFeature.Faces.Item(6)

    ' Create a new sketch on the top face.
    Dim oFoldLineSketch As PlanarSketch
    Set oFoldLineSketch = oCompDef.Sketches.Add(oFrontFace)

    ' The end points of the sketch line must lie on an edge

    Dim oEdge1MidPoint As Point
    Set oEdge1MidPoint = oFrontFace.Edges(1).Geometry.MidPoint

    Dim oSketchPoint1 As Point2d
    Set oSketchPoint1 = oFoldLineSketch.ModelToSketchSpace(oEdge1MidPoint)

    Dim oEdge2MidPoint As Point
    Set oEdge2MidPoint = oFrontFace.Edges(3).Geometry.MidPoint

    Dim oSketchPoint2 As Point2d
    Set oSketchPoint2 = oFoldLineSketch.ModelToSketchSpace(oEdge2MidPoint)

    ' Create the fold line between the midpoint of two opposite edges on the face
    Dim oFoldLine As SketchLine
    Set oFoldLine = oFoldLineSketch.SketchLines.AddByTwoPoints(oSketchPoint1, oSketchPoint2)

    Dim oFoldDefinition As FoldDefinition
    Set oFoldDefinition = oSheetMetalFeatures.FoldFeatures.CreateFoldDefinition(oFoldLine, "60 deg")

    ' Create a fold feature
    Dim oFoldFeature As FoldFeature
    Set oFoldFeature = oSheetMetalFeatures.FoldFeatures.Add(oFoldDefinition)

    ThisApplication.ActiveView.GoHome

End Sub
```

Copy Code

```
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
```

```
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the new sketch.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    oFrontFace = oFaceFeature.Faces.Item(6)

    ' Create a new sketch on the top face.
    Dim oFoldLineSketch As PlanarSketch
    oFoldLineSketch = oCompDef.Sketches.Add(oFrontFace)

    ' The end points of the sketch line must lie on an edge

    Dim oEdge1MidPoint As Point
    oEdge1MidPoint = oFrontFace.Edges(1).Geometry.MidPoint

    Dim oSketchPoint1 As Point2d
    oSketchPoint1 = oFoldLineSketch.ModelToSketchSpace(oEdge1MidPoint)

    Dim oEdge2MidPoint As Point
    oEdge2MidPoint = oFrontFace.Edges(3).Geometry.MidPoint

    Dim oSketchPoint2 As Point2d
    oSketchPoint2 = oFoldLineSketch.ModelToSketchSpace(oEdge2MidPoint)

    ' Create the fold line between the midpoint of two opposite edges on the face
    Dim oFoldLine As SketchLine
    oFoldLine = oFoldLineSketch.SketchLines.AddByTwoPoints(oSketchPoint1, oSketchPoint2)

    Dim oFoldDefinition As FoldDefinition
    oFoldDefinition = oSheetMetalFeatures.FoldFeatures.CreateFoldDefinition(oFoldLine, "60 deg")

    ' Create a fold feature
    Dim oFoldFeature As FoldFeature
    oFoldFeature = oSheetMetalFeatures.FoldFeatures.Add(oFoldDefinition)

    ThisApplication.ActiveView.GoHome
```

Create sheet metal lofted flange feature

Description

The following sample demonstrates the creation of a sheet metal lofted flange feature.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub LoftedFeatureCreation()
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    Set oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    Set oSheetMetalFeatures = oCompDef.Features

    ' Set a reference to the transient geometry object.
```

```
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create the first sketch on the X-Y work plane.
    Dim oSketch1 As PlanarSketch
    Set oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oSketchLines As SketchEntitiesEnumerator
    Set oSketchLines = oSketch1.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(-2, -2), _
                               oTransGeom.CreatePoint2d(2, 2))

    ' Create a path.
    Dim oPath1 As Path
    Set oPath1 = oCompDef.Features.CreatePath(oSketchLines.Item(1))

    ' Create a workplane offset from the X-Y plane by 2 inches.
    Dim oWorkPlane As WorkPlane
    Set oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes.Item(3), "2 in")
    oWorkPlane.Visible = False

    ' Create the second sketch on the new work plane.
    Dim oSketch2 As PlanarSketch
    Set oSketch2 = oCompDef.Sketches.Add(oWorkPlane)

    ' Draw a circle with center at (0,0) and radius 5.
    Dim oSketchCircle As SketchCircle
    Set oSketchCircle = oSketch2.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), 5)

    ' Create a path.
    Dim oPath2 As Path
    Set oPath2 = oCompDef.Features.CreatePath(oSketchCircle)

    Dim oLoftedFlangeDefinition As LoftedFlangeDefinition
    Set oLoftedFlangeDefinition = oSheetMetalFeatures.LoftedFlangeFeatures.CreateLoftedFlangeDefinition(oPath1, oPath2)

    ' Set the output type to press brake with a chord tolerance of .1
    Call oLoftedFlangeDefinition.SetOutputType(kPressBrakeChordToleranceLoftedFlange, "0.1 in")

    ' Create a lofted flange feature.
    Dim oLoftedFlangeFeature As LoftedFlangeFeature
    Set oLoftedFlangeFeature = oSheetMetalFeatures.LoftedFlangeFeatures.Add(oLoftedFlangeDefinition)

    ThisApplication.ActiveView.GoHome
End Sub
```

Copy Code

```
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    oSheetMetalFeatures = oCompDef.Features

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Create the first sketch on the X-Y work plane.
    Dim oSketch1 As PlanarSketch
    oSketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oSketchLines As SketchEntitiesEnumerator
    oSketchLines = oSketch1.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(-2, -2), _
                               oTransGeom.CreatePoint2d(2, 2))

    ' Create a path.
    Dim oPath1 As Path
    oPath1 = oCompDef.Features.CreatePath(oSketchLines.Item(1))

    ' Create a workplane offset from the X-Y plane by 2 inches.
    Dim oWorkPlane As WorkPlane
    oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes.Item(3), "2 in")
    oWorkPlane.Visible = False

    ' Create the second sketch on the new work plane.
    Dim oSketch2 As PlanarSketch
    oSketch2 = oCompDef.Sketches.Add(oWorkPlane)

    ' Draw a circle with center at (0,0) and radius 5.
    Dim oSketchCircle As SketchCircle
    oSketchCircle = oSketch2.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), 5)

    ' Create a path.
    Dim oPath2 As Path
    oPath2 = oCompDef.Features.CreatePath(oSketchCircle)

    Dim oLoftedFlangeDefinition As LoftedFlangeDefinition
    oLoftedFlangeDefinition = oSheetMetalFeatures.LoftedFlangeFeatures.CreateLoftedFlangeDefinition(oPath1, oPath2)

    ' Set the output type to press brake with a chord tolerance of .1
    Call oLoftedFlangeDefinition.SetOutputType(kPressBrakeChordToleranceLoftedFlange, "0.1 in")

    ' Create a lofted flange feature.
    Dim oLoftedFlangeFeature As LoftedFlangeFeature
    oLoftedFlangeFeature = oSheetMetalFeatures.LoftedFlangeFeatures.Add(oLoftedFlangeDefinition)
```

```
    ThisApplication.ActiveView.GoHome
```

Report on punch feature ID's

Description

This program demonstrates accessing punch features and creates a report of the punch ID's used.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PunchFeatureReport()

    ' Get the active sheet metal document and component
    ' definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    If Err Then
        MsgBox "A part must be active."
        Exit Sub
    End If

    Dim oSMDef As SheetMetalComponentDefinition
    Set oSMDef = oPartDoc.ComponentDefinition

    Dim oSMFeatures As SheetMetalFeatures
    Set oSMFeatures = oSMDef.Features

    Dim iUniquePunchCount As Integer
    iUniquePunchCount = 0
    Dim astrPunchIDs() As String
    ReDim astrPunchIDs(1, oSMFeatures.PunchToolFeatures.Count)

    Dim oPunchFeature As PunchToolFeature
    For Each oPunchFeature In oSMFeatures.PunchToolFeatures
        ' Check to see if this punch ID is already in the list.
        Dim bPunchFound As Boolean
        bPunchFound = False
        Dim i As Integer
        For i = 1 To iUniquePunchCount
            If astrPunchIDs(0, i - 1) = oPunchFeature.PunchId Then
                ' Increment the count for this punch ID.
                astrPunchIDs(1, i - 1) = astrPunchIDs(1, i - 1) + 1

                bPunchFound = True
                Exit For
            End If
        Next

        If Not bPunchFound Then
            ' Add this punch to the list.
            iUniquePunchCount = iUniquePunchCount + 1
            astrPunchIDs(0, iUniquePunchCount - 1) = oPunchFeature.PunchId
            astrPunchIDs(1, iUniquePunchCount - 1) = 1
        End If
    Next

    Dim strMessage As String
    strMessage = "Punch Report (Name, Quantity):" & vbCr & vbCr
    For i = 1 To iUniquePunchCount
       strMessage = strMessage & "    " & astrPunchIDs(0, i - 1) & ", " & _
 astrPunchIDs(1, i - 1)
    Next

    MsgBox strMessage
End Sub
```

Copy Code

```
    ' Get the active sheet metal document and component
    ' definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    If Err.Number Then
        MsgBox("A part must be active.")
        Exit Sub
    End If

    Dim oSMDef As SheetMetalComponentDefinition
    oSMDef = oPartDoc.ComponentDefinition

    Dim oSMFeatures As SheetMetalFeatures
    oSMFeatures = oSMDef.Features

    Dim iUniquePunchCount As Integer
    iUniquePunchCount = 0
    Dim astrPunchIDs(1, oSMFeatures.PunchToolFeatures.Count) As String

    Dim oPunchFeature As PunchToolFeature
    For Each oPunchFeature In oSMFeatures.PunchToolFeatures
```

```
        ' Check to see if this punch ID is already in the list.
        Dim bPunchFound As Boolean
        bPunchFound = False
        Dim i As Integer
        For i = 1 To iUniquePunchCount
            If astrPunchIDs(0, i - 1) = oPunchFeature.PunchId Then
                ' Increment the count for this punch ID.
                astrPunchIDs(1, i - 1) = astrPunchIDs(1, i - 1) + 1

                bPunchFound = True
                Exit For
            End If
        Next

        If Not bPunchFound Then
            ' Add this punch to the list.
            iUniquePunchCount = iUniquePunchCount + 1
            astrPunchIDs(0, iUniquePunchCount - 1) = oPunchFeature.PunchId
            astrPunchIDs(1, iUniquePunchCount - 1) = 1
        End If
    Next

    Dim strMessage As String
    strMessage = "Punch Report (Name, Quantity):" & vbCr & vbCr
    For i = 1 To iUniquePunchCount
       strMessage = strMessage & "   " & astrPunchIDs(0, i - 1) & ", " & _
                    astrPunchIDs(1, i - 1)
    Next

    MsgBox(strMessage)
```

Add a punch tool feature

Description

This program demonstrates the creation of a punch tool feature. It uses one of the punch features that's delivered with Inventor. It assumes you already have an existing sheet metal part and have selected a face to place the punch feature on. The selected face should be large so there is room for the punch features.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PlacePunchFeature()
    ' Get the active sheet metal document and component
    ' definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument
    If Err Then
        MsgBox "A part must be active."
        Exit Sub
    End If
    Dim oSMDef As SheetMetalComponentDefinition
    Set oSMDef = oPartDoc.ComponentDefinition

    ' Get the selected face that will be used for the creation
    ' of the sketch that will contain the sketch points.
    Dim oFace As Face
    Set oFace = oPartDoc.SelectSet.Item(1)
    If Err Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType  kPlaneSurface Then
        MsgBox "A planar face must be selected."
        Exit Sub
    End If

    ' Create a sketch on the selected face.
    Dim oSketch As PlanarSketch
    Set oSketch = oSMDef.Sketches.Add(oFace)

    ' Create some points on the sketch.  The model will need to
    ' be of a size that these points lie on the model.
    Dim oPoints As ObjectCollection
    Set oPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oPoint As SketchPoint
    Set oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(2, 2), True)
    Call oPoints.Add(oPoint)

    Set oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(8, 3), True)
    Call oPoints.Add(oPoint)

    Set oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(2, 10), False)
    Call oPoints.Add(oPoint)
    Dim oSMFeatures As SheetMetalFeatures
    Set oSMFeatures = oSMDef.Features

    ' Create an iFeatureDefinition object for a punch tool.
```

```
    Dim oiFeatureDef As iFeatureDefinition
    Set oiFeatureDef = oSMFeatures.PunchToolFeatures.CreateiFeatureDefinition( _
        "C:\Users\Public\Documents\Autodesk\Inventor 2026\Catalog\Punches\keyhole.ide")

    ' Set the input.
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Length"
                Set oParamInput = oInput
                oParamInput.Expression = "1 in"
            Case "Hole_Diameter"
                Set oParamInput = oInput
                oParamInput.Expression = "0.5 in"
            Case "Slot_Width"
                Set oParamInput = oInput
                oParamInput.Expression = "0.3875 in"
            Case "Fillet"
                Set oParamInput = oInput
                oParamInput.Expression = "0.0625 in"
            Case "Thickness"
                Set oParamInput = oInput
                oParamInput.Expression = "0.125 in"
        End Select
    Next

    ' Create the iFeature at a 45 degree angle.
    Dim oPunchTool As PunchToolFeature
    Set oPunchTool = oSMFeatures.PunchToolFeatures.Add(oPoints, oiFeatureDef, 3.14159265358979 / 4)
End Sub
```

Copy Code

```
    ' Get the active sheet metal document and component
    ' definition of the active document.
    On Error Resume Next
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument
    If Err.Number > 0 Then
        MsgBox("A part must be active.")
        Exit Sub
    End If
    Dim oSMDef As SheetMetalComponentDefinition
    oSMDef = oPartDoc.ComponentDefinition

    ' Get the selected face that will be used for the creation
    ' of the sketch that will contain the sketch points.
    Dim oFace As Face
    oFace = oPartDoc.SelectSet.Item(1)
    If Err.Number > 0 Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If
    On Error GoTo 0

    If oFace.SurfaceType <> kPlaneSurface Then
        MsgBox("A planar face must be selected.")
        Exit Sub
    End If

    ' Create a sketch on the selected face.
    Dim oSketch As PlanarSketch
    oSketch = oSMDef.Sketches.Add(oFace)

    ' Create some points on the sketch.  The model will need to
    ' be of a size that these points lie on the model.
    Dim oPoints As ObjectCollection
    oPoints = ThisApplication.TransientObjects.CreateObjectCollection

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oPoint As SketchPoint
    oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(2, 2), True)
    Call oPoints.Add(oPoint)

    oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(8, 3), True)
    Call oPoints.Add(oPoint)

    oPoint = oSketch.SketchPoints.Add(oTG.CreatePoint2d(2, 10), False)
    Call oPoints.Add(oPoint)
    Dim oSMFeatures As SheetMetalFeatures
    oSMFeatures = oSMDef.Features

    ' Create an iFeatureDefinition object for a punch tool.
    Dim oiFeatureDef As iFeatureDefinition
    oiFeatureDef = oSMFeatures.PunchToolFeatures.CreateiFeatureDefinition( _
        "C:\Users\Public\Documents\Autodesk\Inventor 2026\Catalog\Punches\keyhole.ide")

    ' Set the input.
    Dim oInput As iFeatureInput
    For Each oInput In oiFeatureDef.iFeatureInputs
        Dim oParamInput As iFeatureParameterInput
        Select Case oInput.Name
            Case "Length"
                oParamInput = oInput
                oParamInput.Expression = "1 in"
            Case "Hole_Diameter"
                oParamInput = oInput
                oParamInput.Expression = "0.5 in"
            Case "Slot_Width"
                oParamInput = oInput
                oParamInput.Expression = "0.3875 in"
            Case "Fillet"
                oParamInput = oInput
                oParamInput.Expression = "0.0625 in"
```

```
            Case "Thickness"
                oParamInput = oInput
                oParamInput.Expression = "0.125 in"
        End Select
    Next

    ' Create the iFeature at a 45 degree angle.
    Dim oPunchTool As PunchToolFeature
    oPunchTool = oSMFeatures.PunchToolFeatures.Add(oPoints, oiFeatureDef, 3.14159265358979 / 4)
```

Create sheet metal rip feature

Description

This sample demonstrates the creation of a rip sheet metal feature.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub RipFeatureCreation()
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    Set oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    Set oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 2cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 2))

    Dim oSketchLines As ObjectCollection
    Set oSketchLines = ThisApplication.TransientObjects.CreateObjectCollection

    oSketchLines.Add oSketch.SketchLines(1)

    Call oSketch.OffsetSketchEntitiesUsingDistance(oSketchLines, oCompDef.Thickness.Value, True, True)

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrude feature.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent("1 in", kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create bends at all concave edges.
    Dim oEdgeColl As EdgeCollection
    Set oEdgeColl = oExtrude.SurfaceBodies(1).ConcaveEdges

    Dim oBendEdge As Edge
    For Each oBendEdge In oEdgeColl

        Dim oTempColl As EdgeCollection
        Set oTempColl = ThisApplication.TransientObjects.CreateEdgeCollection

        oTempColl.Add oBendEdge

        Dim oBendDef As BendDefinition
        Set oBendDef = oSheetMetalFeatures.BendFeatures.CreateBendDefinition(oTempColl)

        Dim oBendFeature As BendFeature
        Set oBendFeature = oSheetMetalFeatures.BendFeatures.Add(oBendDef)

    Next

    ' Get the first side face of the extrude
    Dim oSideFace As Face
    Set oSideFace = oExtrude.SideFaces.Item(1)

    ' Get any edge on the face that is parallel to the sketch plane
    Dim oEdge As Edge
    For Each oEdge In oSideFace.Edges

        Dim oLineSeg As LineSegment
        Set oLineSeg = oEdge.Geometry
```

```
        Dim oLine As Line
        Set oLine = oTransGeom.CreateLine(oLineSeg.StartPoint, oLineSeg.Direction.AsVector)

        If oSketch.PlanarEntityGeometry.IsParallelTo(oLine) Then
            Exit For
        End If
    Next

    ' Create a workpoint at the edge mid-point
    Dim oWorkPoint As WorkPoint
    Set oWorkPoint = oCompDef.WorkPoints.AddByMidPoint(oEdge)

    ' Create a single point type rip feature
    Dim oRipDef As RipDefinition
    Set oRipDef = oSheetMetalFeatures.RipFeatures.CreateRipDefinition(oSideFace)

    Call oRipDef.SetSinglePointRipType(oSideFace, oWorkPoint, oCompDef.GapSize.Value, kSymmetricExtentDirection)

    Dim oRipFeature As RipFeature
    Set oRipFeature = oSheetMetalFeatures.RipFeatures.Add(oRipDef)

    ThisApplication.ActiveView.GoHome
End Sub
```

Copy Code

```
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 2cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                                oTransGeom.CreatePoint2d(0, 0), _
                                oTransGeom.CreatePoint2d(4, 2))

    Dim oSketchLines As ObjectCollection
    oSketchLines = ThisApplication.TransientObjects.CreateObjectCollection

    oSketchLines.Add(oSketch.SketchLines(1))

    Call oSketch.OffsetSketchEntitiesUsingDistance(oSketchLines, oCompDef.Thickness.Value, True, True)

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create an extrude feature.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent("1 in", kPositiveExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    ' Create bends at all concave edges.
    Dim oEdgeColl As EdgeCollection
    oEdgeColl = oExtrude.SurfaceBodies(1).ConcaveEdges

    Dim oBendEdge As Edge
    For Each oBendEdge In oEdgeColl

        Dim oTempColl As EdgeCollection
        oTempColl = ThisApplication.TransientObjects.CreateEdgeCollection

        oTempColl.Add(oBendEdge)

        Dim oBendDef As BendDefinition
        oBendDef = oSheetMetalFeatures.BendFeatures.CreateBendDefinition(oTempColl)

        Dim oBendFeature As BendFeature
        oBendFeature = oSheetMetalFeatures.BendFeatures.Add(oBendDef)
    Next

    ' Get the first side face of the extrude
    Dim oSideFace As Face
    oSideFace = oExtrude.SideFaces.Item(1)

    ' Get any edge on the face that is parallel to the sketch plane
    Dim oEdge As Edge
    For Each oEdge In oSideFace.Edges

        Dim oLineSeg As LineSegment
        oLineSeg = oEdge.Geometry

        Dim oLine As Line
        oLine = oTransGeom.CreateLine(oLineSeg.StartPoint, oLineSeg.Direction.AsVector)

        If oSketch.PlanarEntityGeometry.IsParallelTo(oLine) Then
            Exit For
        End If
```

```
    Next

    ' Create a workpoint at the edge mid-point
    Dim oWorkPoint As WorkPoint
    oWorkPoint = oCompDef.WorkPoints.AddByMidPoint(oEdge)

    ' Create a single point type rip feature
    Dim oRipDef As RipDefinition
    oRipDef = oSheetMetalFeatures.RipFeatures.CreateRipDefinition(oSideFace)

    Call oRipDef.SetSinglePointRipType(oSideFace, oWorkPoint, oCompDef.GapSize.Value, kSymmetricExtentDirection)

    Dim oRipFeature As RipFeature
    oRipFeature = oSheetMetalFeatures.RipFeatures.Add(oRipDef)

    ThisApplication.ActiveView.GoHome
```

Sheet Metal Feature Display

Description

This sample illustrates getting basic information from the various sheet metal features.

Code Samples

- VBA
- iLogic

Before running the sample, open a sheet metal document that contains some features.

Copy Code

```
Public Sub SheetMetalFeatureDisplay()
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox "A sheet metal document must be open."
        Exit Sub
    End If

    ' Get the sheet metal component definition.  Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    Set oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Iterate through the features looking specifically for sheet metal features.
    Dim oFeature As PartFeature
    For Each oFeature In oSheetMetalCompDef.Features
        Select Case oFeature.Type
            Case kFaceFeatureObject
                Dim oFaceFeature As FaceFeature
                Set oFaceFeature = oFeature
                Debug.Print "Face Feature: " & oFaceFeature.Name
                Debug.Print "   Adaptive: " & oFaceFeature.Adaptive
                Debug.Print "   Face Count: " & oFaceFeature.Faces.Count
                Debug.Print "   HealthStatus: " & oFaceFeature.HealthStatus
                Debug.Print "   RangeBox: (" & Format(oFaceFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                                Format(oFaceFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                                Format(oFaceFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                                Format(oFaceFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                                Format(oFaceFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                                Format(oFaceFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
                Debug.Print "   Suppressed: " & oFaceFeature.Suppressed
            Case kContourFlangeFeatureObject
                Dim oContourFlangeFeature As ContourFlangeFeature
                Set oContourFlangeFeature = oFeature
                Debug.Print "Contour Flange Feature: " & oContourFlangeFeature.Name
                Debug.Print "   Adaptive: " & oContourFlangeFeature.Adaptive
                Debug.Print "   Face Count: " & oContourFlangeFeature.Faces.Count
                Debug.Print "   HealthStatus: " & oContourFlangeFeature.HealthStatus
                Debug.Print "   RangeBox: (" & Format(oContourFlangeFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                                Format(oContourFlangeFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                                Format(oContourFlangeFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                                Format(oContourFlangeFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                                Format(oContourFlangeFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                                Format(oContourFlangeFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
                Debug.Print "   Suppressed: " & oContourFlangeFeature.Suppressed
            Case kCutFeatureObject
                Dim oCutFeature As CutFeature
                Set oCutFeature = oFeature
                Debug.Print "Cut Feature: " & oCutFeature.Name
                Debug.Print "   Adaptive: " & oCutFeature.Adaptive
                Debug.Print "   Face Count: " & oCutFeature.Faces.Count
                Debug.Print "   HealthStatus: " & oCutFeature.HealthStatus
                Debug.Print "   RangeBox: (" & Format(oCutFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                                Format(oCutFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                                Format(oCutFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                                Format(oCutFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                                Format(oCutFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                                Format(oCutFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
                Debug.Print "   Suppressed: " & oCutFeature.Suppressed
            Case kFlangeFeatureObject
                Dim oFlangeFeature As FlangeFeature
                Set oFlangeFeature = oFeature
```

```
        Debug.Print "Flange Feature: " & oFlangeFeature.Name
        Debug.Print "   Adaptive: " & oFlangeFeature.Adaptive
        Debug.Print "   Face Count: " & oFlangeFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oFlangeFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oFlangeFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oFlangeFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oFlangeFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oFlangeFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oFlangeFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oFlangeFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oFlangeFeature.Suppressed
    Case kHemFeatureObject
        Dim oHemFeature As HemFeature
        Set oHemFeature = oFeature
        Debug.Print "Hem Feature: " & oHemFeature.Name
        Debug.Print "   Adaptive: " & oHemFeature.Adaptive
        Debug.Print "   Face Count: " & oHemFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oHemFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oHemFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oHemFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oHemFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oHemFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oHemFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oHemFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oHemFeature.Suppressed
    Case kFoldFeatureObject
        Dim oFoldFeature As FoldFeature
        Set oFoldFeature = oFeature
        Debug.Print "Fold Feature: " & oFoldFeature.Name
        Debug.Print "   Adaptive: " & oFoldFeature.Adaptive
        Debug.Print "   Face Count: " & oFoldFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oFoldFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oFoldFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oFoldFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oFoldFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oFoldFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oFoldFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oFoldFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oFoldFeature.Suppressed
    Case kCornerFeatureObject
        Dim oCornerFeature As CornerFeature
        Set oCornerFeature = oFeature
        Debug.Print "Corner Seam Feature: " & oCornerFeature.Name
        Debug.Print "   Adaptive: " & oCornerFeature.Adaptive
        Debug.Print "   Face Count: " & oCornerFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oCornerFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oCornerFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oCornerFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oCornerFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oCornerFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oCornerFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oCornerFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oCornerFeature.Suppressed
    Case kBendFeatureObject
        Dim oBendFeature As BendFeature
        Set oBendFeature = oFeature
        Debug.Print "Bend Feature: " & oBendFeature.Name
        Debug.Print "   Adaptive: " & oBendFeature.Adaptive
        Debug.Print "   Face Count: " & oBendFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oBendFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oBendFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oBendFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oBendFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oBendFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oBendFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oBendFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oBendFeature.Suppressed
    Case kCornerRoundFeatureObject
        Dim oCornerRoundFeature As CornerRoundFeature
        Set oCornerRoundFeature = oFeature
        Debug.Print "Corner Round Feature: " & oCornerRoundFeature.Name
        Debug.Print "   Adaptive: " & oCornerRoundFeature.Adaptive
        Debug.Print "   Face Count: " & oCornerRoundFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oCornerRoundFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oCornerRoundFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oCornerRoundFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oCornerRoundFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oCornerRoundFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oCornerRoundFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oCornerRoundFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oCornerRoundFeature.Suppressed
    Case kCornerChamferFeatureObject
        Dim oCornerChamferFeature As CornerChamferFeature
        Set oCornerChamferFeature = oFeature
        Debug.Print "Corner Chamfer Feature: " & oCornerChamferFeature.Name
        Debug.Print "   Adaptive: " & oCornerChamferFeature.Adaptive
        Debug.Print "   Face Count: " & oCornerChamferFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oCornerChamferFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oCornerChamferFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oCornerChamferFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oCornerChamferFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oCornerChamferFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oCornerChamferFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                               Format(oCornerChamferFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
        Debug.Print "   Suppressed: " & oCornerChamferFeature.Suppressed
    Case kPunchToolFeatureObject
        Dim oPunchToolFeature As PunchToolFeature
        Set oPunchToolFeature = oFeature
        Debug.Print "Punch Tool Feature: " & oPunchToolFeature.Name
        Debug.Print "   Adaptive: " & oPunchToolFeature.Adaptive
        Debug.Print "   Face Count: " & oPunchToolFeature.Faces.Count
        Debug.Print "   HealthStatus: " & oPunchToolFeature.HealthStatus
        Debug.Print "   RangeBox: (" & Format(oPunchToolFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                               Format(oPunchToolFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                               Format(oPunchToolFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                               Format(oPunchToolFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                               Format(oPunchToolFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
```

```
                                Format(oPunchToolFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
            Debug.Print "   Suppressed: " & oPunchToolFeature.Suppressed
        Case Else
            Debug.Print "Non Sheetmetal Feature: " & oFeature.Name
            Debug.Print "   Adaptive: " & oFeature.Adaptive
            Debug.Print "   Face Count: " & oFeature.Faces.Count
            Debug.Print "   HealthStatus: " & oFeature.HealthStatus
            Debug.Print "   RangeBox: (" & Format(oFeature.RangeBox.MinPoint.X, "0.00000") & ", " & _
                                Format(oFeature.RangeBox.MinPoint.Y, "0.00000") & ", " & _
                                Format(oFeature.RangeBox.MinPoint.Z, "0.00000") & ")-(" & _
                                Format(oFeature.RangeBox.MaxPoint.X, "0.00000") & ", " & _
                                Format(oFeature.RangeBox.MaxPoint.Y, "0.00000") & ", " & _
                                Format(oFeature.RangeBox.MaxPoint.Z, "0.00000") & ")"
            Debug.Print "   Suppressed: " & oFeature.Suppressed
    End Select
  Next
End Sub
```

Before running the sample, open a sheet metal document that contains some features.

Copy Code

```
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox("A sheet metal document must be open.")
        Exit Sub
    End If

    ' Get the sheet metal component definition.  Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Iterate through the features looking specifically for sheet metal features.
    Dim oFeature As PartFeature
    For Each oFeature In oSheetMetalCompDef.Features
        Select Case oFeature.Type
            Case kFaceFeatureObject
                Dim oFaceFeature As FaceFeature
                oFaceFeature = oFeature
                Logger.Info("Face Feature: " & oFaceFeature.Name)

                Logger.Info("   Adaptive: " & oFaceFeature.Adaptive)

                Logger.Info("   Face Count: " & oFaceFeature.Faces.Count)

                Logger.Info("   HealthStatus: " & oFaceFeature.HealthStatus)
                Logger.Info("   RangeBox: (" & Strings.FormatNumber(oFaceFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oFaceFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oFaceFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oFaceFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oFaceFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oFaceFeature.RangeBox.MaxPoint.Z, 5) & ")")
                Logger.Info("   Suppressed: " & oFaceFeature.Suppressed)

            Case kContourFlangeFeatureObject
                Dim oContourFlangeFeature As ContourFlangeFeature
                oContourFlangeFeature = oFeature
                Logger.Info("Contour Flange Feature: " & oContourFlangeFeature.Name)

                Logger.Info("   Adaptive: " & oContourFlangeFeature.Adaptive)

                Logger.Info("   Face Count: " & oContourFlangeFeature.Faces.Count)

                Logger.Info("   HealthStatus: " & oContourFlangeFeature.HealthStatus)

                Logger.Info("   RangeBox: (" & FormatNumber(oContourFlangeFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oContourFlangeFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oContourFlangeFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oContourFlangeFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oContourFlangeFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oContourFlangeFeature.RangeBox.MaxPoint.Z, 5) & ")")
                Logger.Info("   Suppressed: " & oContourFlangeFeature.Suppressed)

            Case kCutFeatureObject
                Dim oCutFeature As CutFeature
                oCutFeature = oFeature
                Logger.Info("Cut Feature: " & oCutFeature.Name)

                Logger.Info("   Adaptive: " & oCutFeature.Adaptive)

                Logger.Info("   Face Count: " & oCutFeature.Faces.Count)

                Logger.Info("   HealthStatus: " & oCutFeature.HealthStatus)

                Logger.Info("   RangeBox: (" & FormatNumber(oCutFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oCutFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oCutFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oCutFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oCutFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oCutFeature.RangeBox.MaxPoint.Z, 5) & ")")
                Logger.Info("   Suppressed: " & oCutFeature.Suppressed)

            Case kFlangeFeatureObject
                Dim oFlangeFeature As FlangeFeature
                oFlangeFeature = oFeature
                Logger.Info("Flange Feature: " & oFlangeFeature.Name)

                Logger.Info("   Adaptive: " & oFlangeFeature.Adaptive)

                Logger.Info("   Face Count: " & oFlangeFeature.Faces.Count)
```

```
        Logger.Info("   HealthStatus: " & oFlangeFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oFlangeFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oFlangeFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oFlangeFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oFlangeFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oFlangeFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oFlangeFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oFlangeFeature.Suppressed)

    Case kHemFeatureObject
        Dim oHemFeature As HemFeature
        oHemFeature = oFeature
        Logger.Info("Hem Feature: " & oHemFeature.Name)

        Logger.Info("   Adaptive: " & oHemFeature.Adaptive)

        Logger.Info("   Face Count: " & oHemFeature.Faces.Count)

        Logger.Info("   HealthStatus: " & oHemFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oHemFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oHemFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oHemFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oHemFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oHemFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oHemFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oHemFeature.Suppressed)

    Case kFoldFeatureObject
        Dim oFoldFeature As FoldFeature
        oFoldFeature = oFeature
        Logger.Info("Fold Feature: " & oFoldFeature.Name)

        Logger.Info("   Adaptive: " & oFoldFeature.Adaptive)

        Logger.Info("   Face Count: " & oFoldFeature.Faces.Count)

        Logger.Info("   HealthStatus: " & oFoldFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oFoldFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oFoldFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oFoldFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oFoldFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oFoldFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oFoldFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oFoldFeature.Suppressed)

    Case kCornerFeatureObject
        Dim oCornerFeature As CornerFeature
        oCornerFeature = oFeature
        Logger.Info("Corner Seam Feature: " & oCornerFeature.Name)

        Logger.Info("   Adaptive: " & oCornerFeature.Adaptive)

        Logger.Info("   Face Count: " & oCornerFeature.Faces.Count)

        Logger.Info("   HealthStatus: " & oCornerFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oCornerFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oCornerFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oCornerFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oCornerFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oCornerFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oCornerFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oCornerFeature.Suppressed)

    Case kBendFeatureObject
        Dim oBendFeature As BendFeature
        oBendFeature = oFeature
        Logger.Info("Bend Feature: " & oBendFeature.Name)

        Logger.Info("   Adaptive: " & oBendFeature.Adaptive)

        Logger.Info("   Face Count: " & oBendFeature.Faces.Count)

        Logger.Info("   HealthStatus: " & oBendFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oBendFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oBendFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oBendFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oBendFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oBendFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oBendFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oBendFeature.Suppressed)

    Case kCornerRoundFeatureObject
        Dim oCornerRoundFeature As CornerRoundFeature
        oCornerRoundFeature = oFeature
        Logger.Info("Corner Round Feature: " & oCornerRoundFeature.Name)

        Logger.Info("   Adaptive: " & oCornerRoundFeature.Adaptive)

        Logger.Info("   Face Count: " & oCornerRoundFeature.Faces.Count)

        Logger.Info("   HealthStatus: " & oCornerRoundFeature.HealthStatus)

        Logger.Info("   RangeBox: (" & FormatNumber(oCornerRoundFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                FormatNumber(oCornerRoundFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                FormatNumber(oCornerRoundFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                FormatNumber(oCornerRoundFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                FormatNumber(oCornerRoundFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                FormatNumber(oCornerRoundFeature.RangeBox.MaxPoint.Z, 5) & ")")
        Logger.Info("   Suppressed: " & oCornerRoundFeature.Suppressed)

    Case kCornerChamferFeatureObject
        Dim oCornerChamferFeature As CornerChamferFeature
```

```
            oCornerChamferFeature = oFeature
            Logger.Info("Corner Chamfer Feature: " & oCornerChamferFeature.Name)

            Logger.Info("   Adaptive: " & oCornerChamferFeature.Adaptive)

            Logger.Info("   Face Count: " & oCornerChamferFeature.Faces.Count)

            Logger.Info("   HealthStatus: " & oCornerChamferFeature.HealthStatus)

            Logger.Info("   RangeBox: (" & FormatNumber(oCornerChamferFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                    FormatNumber(oCornerChamferFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                    FormatNumber(oCornerChamferFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                    FormatNumber(oCornerChamferFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                    FormatNumber(oCornerChamferFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                    FormatNumber(oCornerChamferFeature.RangeBox.MaxPoint.Z, 5) & ")")
            Logger.Info("   Suppressed: " & oCornerChamferFeature.Suppressed)

        Case kPunchToolFeatureObject
            Dim oPunchToolFeature As PunchToolFeature
            oPunchToolFeature = oFeature
            Logger.Info("Punch Tool Feature: " & oPunchToolFeature.Name)

            Logger.Info("   Adaptive: " & oPunchToolFeature.Adaptive)

            Logger.Info("   Face Count: " & oPunchToolFeature.Faces.Count)

            Logger.Info("   HealthStatus: " & oPunchToolFeature.HealthStatus)

            Logger.Info("   RangeBox: (" & FormatNumber(oPunchToolFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                    FormatNumber(oPunchToolFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                    FormatNumber(oPunchToolFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                    FormatNumber(oPunchToolFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                    FormatNumber(oPunchToolFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                    FormatNumber(oPunchToolFeature.RangeBox.MaxPoint.Z, 5) & ")")
            Logger.Info("   Suppressed: " & oPunchToolFeature.Suppressed)

        Case Else
            Logger.Info("Non Sheetmetal Feature: " & oFeature.Name)

            Logger.Info("   Adaptive: " & oFeature.Adaptive)

            Logger.Info("   Face Count: " & oFeature.Faces.Count)

            Logger.Info("   HealthStatus: " & oFeature.HealthStatus)

            Logger.Info("   RangeBox: (" & FormatNumber(oFeature.RangeBox.MinPoint.X, 5) & ", " & _
                                    FormatNumber(oFeature.RangeBox.MinPoint.Y, 5) & ", " & _
                                    FormatNumber(oFeature.RangeBox.MinPoint.Z, 5) & ")-(" & _
                                    FormatNumber(oFeature.RangeBox.MaxPoint.X, 5) & ", " & _
                                    FormatNumber(oFeature.RangeBox.MaxPoint.Y, 5) & ", " & _
                                    FormatNumber(oFeature.RangeBox.MaxPoint.Z, 5) & ")")
            Logger.Info("   Suppressed: " & oFeature.Suppressed)

    End Select
Next
```

Create sheet metal face and cut features

Description

This sample demonstrates the creation of sheet metal face and cut features.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub FaceAndCutFeatureCreation()
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    Set oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    Set oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
```

```
    Set oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    Set oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    Set oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the new sketch.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    Set oFrontFace = oFaceFeature.Faces.Item(6)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    Set oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (1,0.75,0) is.
    Dim oCorner As Point2d
    Set oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(1, 0.75, 0))

    ' Create the interior 3cm x 2cm rectangle for the cut.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                oCorner, oTransGeom.CreatePoint2d(oCorner.X + 2, oCorner.Y + 1.5))

    ' Create a profile.
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a cut definition object
    Dim oCutDefinition As CutDefinition
    Set oCutDefinition = oSheetMetalFeatures.CutFeatures.CreateCutDefinition(oProfile)

    ' Set extents to 'Through All'
    Call oCutDefinition.SetThroughAllExtent(kNegativeExtentDirection)

    ' Create the cut feature
    Dim oCutFeature As CutFeature
    Set oCutFeature = oSheetMetalFeatures.CutFeatures.Add(oCutDefinition)
End Sub
```

Copy Code

```
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                 ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the new sketch.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    oFrontFace = oFaceFeature.Faces.Item(6)

    ' Create a new sketch on this face, but use the method that allows you to
    ' control the orientation and orgin of the new sketch.
    oSketch = oCompDef.Sketches.AddWithOrientation(oFrontFace, _
                    oCompDef.WorkAxes.Item(1), True, True, oCompDef.WorkPoints(1))

    ' Determine where in sketch space the point (1,0.75,0) is.
    Dim oCorner As Point2d
    oCorner = oSketch.ModelToSketchSpace(oTransGeom.CreatePoint(1, 0.75, 0))

    ' Create the interior 3cm x 2cm rectangle for the cut.
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                oCorner, oTransGeom.CreatePoint2d(oCorner.X + 2, oCorner.Y + 1.5))

    ' Create a profile.
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a cut definition object
    Dim oCutDefinition As CutDefinition
    oCutDefinition = oSheetMetalFeatures.CutFeatures.CreateCutDefinition(oProfile)

    ' Set extents to 'Through All'
```

```
    Call oCutDefinition.SetThroughAllExtent(kNegativeExtentDirection)

    ' Create the cut feature
    Dim oCutFeature As CutFeature
    oCutFeature = oSheetMetalFeatures.CutFeatures.Add(oCutDefinition)
```

Create sheet metal face and flange features

Description

This sample demonstrates the creation of sheet metal face and flange features.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub FaceAndFlangeFeatureCreation()
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    Set oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    Set oSheetMetalFeatures = oCompDef.Features

    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    Set oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    Set oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the flange.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    Set oFrontFace = oFaceFeature.Faces.Item(6)

    ' Collect up all edges of the face
    Dim oEdgeCollection As EdgeCollection
    Set oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oEdge As Edge
    For Each oEdge In oFrontFace.Edges
        Call oEdgeCollection.Add(oEdge)
    Next

    Dim oFlangeDefinition As FlangeDefinition
    Set oFlangeDefinition = oSheetMetalFeatures.FlangeFeatures.CreateFlangeDefinition(oEdgeCollection, 3.14159 / 2, "2 in")

    ' Set the bend radius value
    oFlangeDefinition.BendRadius = oCompDef.BendRadius.Value * 2

    ' Create a flange feature
    Dim oFlangeFeature As FlangeFeature
    Set oFlangeFeature = oSheetMetalFeatures.FlangeFeatures.Add(oFlangeDefinition)

End Sub
```

Copy Code

```
    ' Create a new sheet metal document, using the default sheet metal template.
    Dim oSheetMetalDoc As PartDocument
    oSheetMetalDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject, , , "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}"))

    ' Set a reference to the component definition.
    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oSheetMetalDoc.ComponentDefinition

    ' Set a reference to the sheet metal features collection.
    Dim oSheetMetalFeatures As SheetMetalFeatures
    oSheetMetalFeatures = oCompDef.Features
```

```
    ' Create a new sketch on the X-Y work plane.
    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

    ' Set a reference to the transient geometry object.
    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Call oSketch.SketchLines.AddAsTwoPointRectangle( _
                               oTransGeom.CreatePoint2d(0, 0), _
                               oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    Dim oFaceFeatureDefinition As FaceFeatureDefinition
    oFaceFeatureDefinition = oSheetMetalFeatures.FaceFeatures.CreateFaceFeatureDefinition(oProfile)

    ' Create a face feature.
    Dim oFaceFeature As FaceFeature
    oFaceFeature = oSheetMetalFeatures.FaceFeatures.Add(oFaceFeatureDefinition)

    ' Get the top face for creating the flange.
    ' We'll assume that the 6th face is the top face.
    Dim oFrontFace As Face
    oFrontFace = oFaceFeature.Faces.Item(6)

    ' Collect up all edges of the face
    Dim oEdgeCollection As EdgeCollection
    oEdgeCollection = ThisApplication.TransientObjects.CreateEdgeCollection

    Dim oEdge As Edge
    For Each oEdge In oFrontFace.Edges
        Call oEdgeCollection.Add(oEdge)
    Next

    Dim oFlangeDefinition As FlangeDefinition
    oFlangeDefinition = oSheetMetalFeatures.FlangeFeatures.CreateFlangeDefinition(oEdgeCollection, 3.14159 / 2, "2 in")

    ' Set the bend radius value
    oFlangeDefinition.BendRadius = oCompDef.BendRadius.Value * 2

    ' Create a flange feature
    Dim oFlangeFeature As FlangeFeature
    oFlangeFeature = oSheetMetalFeatures.FlangeFeatures.Add(oFlangeDefinition)
```

Sheet Metal Style Display

Description

This sample illustrates getting information about sheet metal styles.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SheetMetalStyleDisplay()
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox "A sheet metal document must be open."
        Exit Sub
    End If

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    Set oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Display then name of the active style.
    Debug.Print "Active Sheet Metal Style: " & oSheetMetalCompDef.ActiveSheetMetalStyle.Name

    ' Iterate through the sheet metal styles.
    Debug.Print ""
    Debug.Print "Sheet Metal Styles"
    Dim oStyle As SheetMetalStyle

    On Error Resume Next
    For Each oStyle In oSheetMetalCompDef.SheetMetalStyles
        Debug.Print "  " & oStyle.Name
        Debug.Print "     In Use: " & oStyle.InUse
        Debug.Print "     Up to Date: " & oStyle.UpToDate

        Select Case oStyle.StyleLocation
            Case kBothStyleLocation
                Debug.Print "     Style Location: Both Local and in Library"
            Case kLibraryStyleLocation
```

```
            Debug.Print "     Style Location: Library"
        Case kLocalStyleLocation
            Debug.Print "     Style Location: Local"
    End Select

    Debug.Print "     Bend Radius: " & oStyle.BendRadius
    Debug.Print "     Bend Relief Depth: " & oStyle.BendReliefDepth
    Debug.Print "     Bend Relief Width: " & oStyle.BendReliefWidth

    Select Case oStyle.BendReliefShape
        Case kRoundBendReliefShape
            Debug.Print "     Bend Relief Shape: Round"
        Case kStraightBendReliefShape
            Debug.Print "     Bend Relief Shape: Straight"
    End Select

    Select Case oStyle.BendTransition
        Case kArcBendTransition
            Debug.Print "     Bend Transition: Arc"
        Case kDefaultBendTransition
            Debug.Print "     Bend Transition: Default"
        Case kIntersectionBendTransition
            Debug.Print "     Bend Transition: Intersection"
        Case kNoBendTransition
            Debug.Print "     Bend Transition: None"
        Case kStraightLineBendTransition
            Debug.Print "     Bend Transition: Straight Line"
        Case kTrimToBendBendTransition
            Debug.Print "     Bend Transition: Trim to Bend"
    End Select

    Debug.Print "     Bend Transition Arc Radius: " & oStyle.BendTransitionArcRadius

    Debug.Print "     Corner Relief Size: " & oStyle.CornerReliefSize
    Select Case oStyle.CornerReliefShape
        Case kArcWeldCornerReliefShape
            Debug.Print "     Corner Relief Shape: Arc Weld"
        Case kDefaultCornerReliefShape
            Debug.Print "     Corner Relief Shape: Default"
        Case kFullRoundCornerReliefShape
            Debug.Print "     Corner Relief Shape: Full Round"
        Case kIntersectionCornerReliefShape
            Debug.Print "     Corner Relief Shape: Intersection"
        Case kLinearWeldReliefShape
            Debug.Print "     Corner Relief Shape: Linear Weld"
        Case kNoReplacementCornerReliefShape
            Debug.Print "     Corner Relief Shape: No Replacement"
        Case kRoundCornerReliefShape
            Debug.Print "     Corner Relief Shape: Round"
        Case kRoundWithRadiusCornerReliefShape
            Debug.Print "     Corner Relief Shape: Round with Radius"
        Case kSquareCornerReliefShape
            Debug.Print "     Corner Relief Shape: Square"
        Case kTearCornerReliefShape
            Debug.Print "     Corner Relief Shape: Tear"
        Case kTrimToBendReliefShape
            Debug.Print "     Corner Relief Shape: Trim"
    End Select

    Debug.Print "     Corner Relief Size: " & oStyle.ThreeBendCornerReliefSize
    Select Case oStyle.ThreeBendCornerReliefShape
        Case kArcWeldCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Arc Weld"
        Case kDefaultCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Default"
        Case kFullRoundCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Full Round"
        Case kIntersectionCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Intersection"
        Case kLinearWeldReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Linear Weld"
        Case kNoReplacementCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: No Replacement"
        Case kRoundCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Round"
        Case kRoundWithRadiusCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Round with Radius"
        Case kSquareCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Square"
        Case kTearCornerReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Tear"
        Case kTrimToBendReliefShape
            Debug.Print "     Three Bend Corner Relief Shape: Trim"
    End Select

    Debug.Print "     Material: " & oStyle.Material.Name
    Debug.Print "     Minimum Remnant: " & oStyle.MinimumRemnant
    Debug.Print "     Thickness: " & oStyle.Thickness

    Select Case oStyle.PunchRepresentationType
        Case k2DSketchAndCenterMarkPunchRepresentation
            Debug.Print "     Punch Representation Type: 2D Sketch and Center Mark"
        Case k2DSketchPunchRepresentation
            Debug.Print "     Punch Representation Type: 2D Sketch"
        Case kCentermarkPunchRepresentation
            Debug.Print "     Punch Representation Type: Center Mark"
        Case kDefaultPunchRepresentation
            Debug.Print "     Punch Representation Type: Default"
        Case kFormedFeaturePunchRepresentation
            Debug.Print "     Punch Representation Type: Formed Feature"
    End Select

    Debug.Print "     Unfold Method: " & oStyle.UnfoldMethod.Name
Next

' Display information about the unfold methods.
Debug.Print ""
```

```
    Debug.Print "Unfold Methods"
    Dim oUnfoldMethod As UnfoldMethod
    For Each oUnfoldMethod In oSheetMetalCompDef.UnfoldMethods
        Debug.Print "  " & oUnfoldMethod.Name
        Debug.Print "     In Use: " & oUnfoldMethod.InUse
        Debug.Print "     Up to Date: " & oUnfoldMethod.UpToDate

        Select Case oUnfoldMethod.StyleLocation
            Case kBothStyleLocation
                Debug.Print "     Location: Both Local and in Library"
            Case kLibraryStyleLocation
                Debug.Print "     Location: Library"
            Case kLocalStyleLocation
                Debug.Print "     Location: Local"
        End Select

        Select Case oUnfoldMethod.UnfoldMethodType
            Case kBendTableUnfoldMethod
                Debug.Print "     Unfold Method Type: Bend Table"
            Case kLinearUnfoldMethod
                Debug.Print "     Unfold Method Type: Linear"
        End Select
        Debug.Print "     kFactor: " & oUnfoldMethod.kFactor
    Next
End Sub
```

Copy Code

```
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox("A sheet metal document must be open.")
        Exit Sub
    End If

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Display then name of the active style.
    Logger.Info("Active Sheet Metal Style: " & oSheetMetalCompDef.ActiveSheetMetalStyle.Name)

    ' Iterate through the sheet metal styles.
    Logger.Info("")
    Logger.Info("Sheet Metal Styles")

    Dim oStyle As SheetMetalStyle

    On Error Resume Next
    For Each oStyle In oSheetMetalCompDef.SheetMetalStyles
        Logger.Info("  " & oStyle.Name)
        Logger.Info("     In Use: " & oStyle.InUse)
        Logger.Info("     Up to Date: " & oStyle.UpToDate)

        Select Case oStyle.StyleLocation
            Case kBothStyleLocation
                Logger.Info("     Style Location: Both Local and in Library")

            Case kLibraryStyleLocation
                Logger.Info("     Style Location: Library")

            Case kLocalStyleLocation
                Logger.Info("     Style Location: Local")

        End Select

        Logger.Info("     Bend Radius: " & oStyle.BendRadius)
        Logger.Info("     Bend Relief Depth: " & oStyle.BendReliefDepth)
        Logger.Info("     Bend Relief Width: " & oStyle.BendReliefWidth)

        Select Case oStyle.BendReliefShape
            Case kRoundBendReliefShape
                Logger.Info("     Bend Relief Shape: Round")
            Case kStraightBendReliefShape
                Logger.Info("     Bend Relief Shape: Straight")
        End Select

        Select Case oStyle.BendTransition
            Case kArcBendTransition
                Logger.Info("     Bend Transition: Arc")
            Case kDefaultBendTransition
                Logger.Info("     Bend Transition: Default")
            Case kIntersectionBendTransition
                Logger.Info("     Bend Transition: Intersection")
            Case kNoBendTransition
                Logger.Info("     Bend Transition: None")
            Case kStraightLineBendTransition
                Logger.Info("     Bend Transition: Straight Line")
            Case kTrimToBendBendTransition
                Logger.Info("     Bend Transition: Trim to Bend")
        End Select

        Logger.Info("     Bend Transition Arc Radius: " & oStyle.BendTransitionArcRadius)
        Logger.Info("     Corner Relief Size: " & oStyle.CornerReliefSize)

        Select Case oStyle.CornerReliefShape
            Case kArcWeldCornerReliefShape
                Logger.Info("     Corner Relief Shape: Arc Weld")
            Case kDefaultCornerReliefShape
                Logger.Info("     Corner Relief Shape: Default")
            Case kFullRoundCornerReliefShape
```

```
            Logger.Info("     Corner Relief Shape: Full Round")
        Case kIntersectionCornerReliefShape
            Logger.Info("     Corner Relief Shape: Intersection")
        Case kLinearWeldReliefShape
            Logger.Info("     Corner Relief Shape: Linear Weld")
        Case kNoReplacementCornerReliefShape
            Logger.Info("     Corner Relief Shape: No Replacement")
        Case kRoundCornerReliefShape
            Logger.Info("     Corner Relief Shape: Round")
        Case kRoundWithRadiusCornerReliefShape
            Logger.Info("     Corner Relief Shape: Round with Radius")
        Case kSquareCornerReliefShape
            Logger.Info("     Corner Relief Shape: Square")
        Case kTearCornerReliefShape
            Logger.Info("     Corner Relief Shape: Tear")
        Case kTrimToBendReliefShape
            Logger.Info("     Corner Relief Shape: Trim")
    End Select

    Logger.Info("     Corner Relief Size: " & oStyle.ThreeBendCornerReliefSize)

    Select Case oStyle.ThreeBendCornerReliefShape
        Case kArcWeldCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Arc Weld")
        Case kDefaultCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Default")
        Case kFullRoundCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Full Round")
        Case kIntersectionCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Intersection")
        Case kLinearWeldReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Linear Weld")
        Case kNoReplacementCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: No Replacement")
        Case kRoundCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Round")
        Case kRoundWithRadiusCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Round with Radius")
        Case kSquareCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Square")
        Case kTearCornerReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Tear")
        Case kTrimToBendReliefShape
            Logger.Info("     Three Bend Corner Relief Shape: Trim")
    End Select

    Logger.Info("     Material: " & oStyle.Material.Name)
    Logger.Info("     Minimum Remnant: " & oStyle.MinimumRemnant)
    Logger.Info("     Thickness: " & oStyle.Thickness)

    Select Case oStyle.PunchRepresentationType
        Case k2DSketchAndCenterMarkPunchRepresentation
            Logger.Info("     Punch Representation Type: 2D Sketch and Center Mark")
        Case k2DSketchPunchRepresentation
            Logger.Info("     Punch Representation Type: 2D Sketch")
        Case kCentermarkPunchRepresentation
            Logger.Info("     Punch Representation Type: Center Mark")
        Case kDefaultPunchRepresentation
            Logger.Info("     Punch Representation Type: Default")
        Case kFormedFeaturePunchRepresentation
            Logger.Info("     Punch Representation Type: Formed Feature")
    End Select

    Logger.Info("     Unfold Method: " & oStyle.UnfoldMethod.Name)
Next

' Display information about the unfold methods.
Logger.Info("")
Logger.Info("Unfold Methods")

Dim oUnfoldMethod As UnfoldMethod
For Each oUnfoldMethod In oSheetMetalCompDef.UnfoldMethods
    Logger.Info("  " & oUnfoldMethod.Name)
    Logger.Info("     In Use: " & oUnfoldMethod.InUse)
    Logger.Info("     Up to Date: " & oUnfoldMethod.UpToDate)

    Select Case oUnfoldMethod.StyleLocation
        Case kBothStyleLocation
            Logger.Info("     Location: Both Local and in Library")
        Case kLibraryStyleLocation
            Logger.Info("     Location: Library")
        Case kLocalStyleLocation
            Logger.Info("     Location: Local")
    End Select

    Select Case oUnfoldMethod.UnfoldMethodType
        Case kBendTableUnfoldMethod
            Logger.Info("     Unfold Method Type: Bend Table")
        Case kLinearUnfoldMethod
            Logger.Info("     Unfold Method Type: Linear")
    End Select
    Logger.Info("     kFactor: " & oUnfoldMethod.kFactor)
Next
```

Sheet Metal Thickness Editing

Description

This sample illustrates editing the thickness of a sheet metal part.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub SetSheetMetalThickness()
    ' Set a reference to the sheet metal document.
    ' This assumes a sheet metal document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    Set oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Override the thickness for the document
    oSheetMetalCompDef.UseSheetMetalStyleThickness = False

    ' Get a reference to the parameter controlling the thickness.
    Dim oThicknessParam As Parameter
    Set oThicknessParam = oSheetMetalCompDef.Thickness

    ' Change the value of the parameter.
    oThicknessParam.Value = oThicknessParam.Value * 1.5

    ' Update the part.
    ThisApplication.ActiveDocument.Update
End Sub
```

Copy Code

```
    ' Set a reference to the sheet metal document.
    ' This assumes a sheet metal document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Override the thickness for the document
    oSheetMetalCompDef.UseSheetMetalStyleThickness = False

    ' Get a reference to the parameter controlling the thickness.
    Dim oThicknessParam As Parameter
    oThicknessParam = oSheetMetalCompDef.Thickness

    ' Change the value of the parameter.
    oThicknessParam.Value = oThicknessParam.Value * 1.5

    ' Update the part.
    ThisApplication.ActiveDocument.Update
```

Sheet Metal Style Creation

Description

This sample illustrates creating a new sheet metal style. It uses a bend table and assumes the sample bend table delivered with Inventor is available. You can edit the path below to reference any existing bend table. To use the sample make sure a bend table is available at the specified path, open a sheet metal document, and run the sample.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CreateSheetMetalStyle()
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox "A sheet metal document must be open."
        Exit Sub
    End If

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    Set oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Create a new unfold method.
    Dim okFactorUnfoldMethod As UnfoldMethod
```

```
    Set okFactorUnfoldMethod = oSheetMetalCompDef.UnfoldMethods.AddLinearUnfoldMethod("kFactorTest", "0.43")

    ' Create a new unfold method using a bend table.
    Dim oTableUnfoldMethod As UnfoldMethod
    On Error Resume Next
    Set oTableUnfoldMethod = oSheetMetalCompDef.UnfoldMethods.AddBendTableFromFile("Table Sample", "C:\Users\Public\Documents\Autodesk
    If Err Then
        MsgBox "Unable to load bend table"
    End If
    On Error Goto 0

    ' Copy a sheet metal style to create a new one. This arbitrarily uses the
    ' first style in the collection.
    Dim oStyle As SheetMetalStyle
    Set oStyle = oSheetMetalCompDef.SheetMetalStyles.Item(1).Copy("Sample Style")

    ' Set the value for the thickness. This is an expression that will be used to
    ' set a parameter when this style is activated.
    oStyle.Thickness = ".1 in"

    ' The name "Thickness" as used below is handled specially when the style
    ' is used so it will behave as expected for all languages.
    oStyle.BendRadius = "Thickness * 1.5"
    oStyle.BendReliefDepth = "Thickness * 1.5"
    oStyle.BendReliefShape = kRoundBendReliefShape
    oStyle.BendReliefWidth = "Thickness / 2"
    oStyle.BendTransition = kArcBendTransition
    oStyle.BendTransitionArcRadius = "Thickness * 2.0"
    oStyle.CornerReliefShape = kRoundCornerReliefShape
    oStyle.CornerReliefSize = "Thickness * 2.0"
    oStyle.MinimumRemnant = "Thickness * 2.0"

    ' Set the material.  For this example it arbitrarily uses the first material in the collection.
    oStyle.Material = oPartDoc.Materials.Item(1)

    oStyle.PunchRepresentationType = kFormedFeaturePunchRepresentation
    oStyle.ThreeBendCornerReliefShape = kFullRoundCornerReliefShape
    oStyle.ThreeBendCornerReliefSize = "Thickness * 2.0"

    ' Activate this style.
    oStyle.Activate

    ' This section will override some of the settings defined by the style.
    Dim bOverrideStyle As Boolean
    bOverrideStyle = True
    If bOverrideStyle Then
        ' Set the unfold method to use the unfold method created above.
        oSheetMetalCompDef.UnfoldMethod = okFactorUnfoldMethod

        ' Set the thickness.
        oSheetMetalCompDef.UseSheetMetalStyleThickness = False
        oSheetMetalCompDef.Thickness.Expression = ".12 in"

        ' Set the material.
        oSheetMetalCompDef.Material = oPartDoc.Materials.Item(2)
    End If
End Sub
```

Copy Code

```
    ' Set a reference to the sheet metal document.
    ' This assumes a part document is active.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    ' Make sure the document is a sheet metal document.
    If oPartDoc.SubType <> "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}" Then
        MsgBox("A sheet metal document must be open.")
        Exit Sub
    End If

    ' Get the sheet metal component definition. Because this is a part document whose
    ' sub type is sheet metal, the document will return a SheetMetalComponentDefinition
    ' instead of a PartComponentDefinition.
    Dim oSheetMetalCompDef As SheetMetalComponentDefinition
    oSheetMetalCompDef = oPartDoc.ComponentDefinition

    ' Create a new unfold method.
    Dim okFactorUnfoldMethod As UnfoldMethod
    okFactorUnfoldMethod = oSheetMetalCompDef.UnfoldMethods.AddLinearUnfoldMethod("kFactorTest", "0.43")

    ' Create a new unfold method using a bend table.
    Dim oTableUnfoldMethod As UnfoldMethod
    On Error Resume Next
    oTableUnfoldMethod = oSheetMetalCompDef.UnfoldMethods.AddBendTableFromFile("Table Sample", "C:\Users\Public\Documents\Autodesk\Inv
    If Err.Number >0 Then
        MsgBox("Unable to load bend table")
    End If
    On Error Goto 0

    ' Copy a sheet metal style to create a new one. This arbitrarily uses the
    ' first style in the collection.
    Dim oStyle As SheetMetalStyle
    oStyle = oSheetMetalCompDef.SheetMetalStyles.Item(1).Copy("Sample Style")

    ' Set the value for the thickness. This is an expression that will be used to
    ' set a parameter when this style is activated.
    oStyle.Thickness = ".1 in"

    ' The name "Thickness" as used below is handled specially when the style
    ' is used so it will behave as expected for all languages.
    oStyle.BendRadius = "Thickness * 1.5"
    oStyle.BendReliefDepth = "Thickness * 1.5"
    oStyle.BendReliefShape = kRoundBendReliefShape
    oStyle.BendReliefWidth = "Thickness / 2"
    oStyle.BendTransition = kArcBendTransition
    oStyle.BendTransitionArcRadius = "Thickness * 2.0"
```

```
oStyle.CornerReliefShape = kRoundCornerReliefShape
oStyle.CornerReliefSize = "Thickness * 2.0"
oStyle.MinimumRemnant = "Thickness * 2.0"

' Set the material.  For this example it arbitrarily uses the first material in the collection.
oStyle.Material = oPartDoc.Materials.Item(1)

oStyle.PunchRepresentationType = kFormedFeaturePunchRepresentation
oStyle.ThreeBendCornerReliefShape = kFullRoundCornerReliefShape
oStyle.ThreeBendCornerReliefSize = "Thickness * 2.0"

' Activate this style.
oStyle.Activate

' This section will override some of the settings defined by the style.
Dim bOverrideStyle As Boolean
bOverrideStyle = True
If bOverrideStyle Then
    ' Set the unfold method to use the unfold method created above.
    oSheetMetalCompDef.UnfoldMethod = okFactorUnfoldMethod

    ' Set the thickness.
    oSheetMetalCompDef.UseSheetMetalStyleThickness = False
    oSheetMetalCompDef.Thickness.Expression = ".12 in"

    ' Set the material.
    oSheetMetalCompDef.Material = oPartDoc.Materials.Item(2)
End If
```

Translate - Sheet Metal to DXF

Description

The sample code below writes a sheet metal file out as DXF. DWG is also supported. Use either the FLAT PATTERN DWG or FLAT PATTERN DXF formats.There are several optional arguments that can be specified as part of the format string. E.g. 'FLAT PATTERN DXF?TangentLayer=Tangents&SimplifySplines=True'). Below are the names of these arguments and relevant default values. The output will use these values unless you override them as part of the input string.

Argument	Type	Default Value	Note
TangentLayer	String	IV_TANGENT	
OuterProfileLayer	String	IV_OUTER_PROFILE	
ArcCentersLayer	String	IV_ARC_CENTERS	
InteriorProfilesLayer	String	IV_INTERIOR_PROFILES	
BendLayer	String	IV_BEND	BendUpLayer + BendDownLayer (legacy support)
BendUpLayer	String	IV_BEND	
BendDownLayer	String	IV_BEND_DOWN	
ToolCenterLayer	String	IV_TOOL_CENTER	ToolCenterUpLayer + ToolCenterDownLayer (legacy support)
ToolCenterUpLayer	String	IV_TOOL_CENTER	
ToolCenterDownLayer	String	IV_TOOL_CENTER_DOWN	
FeatureProfilesLayer	String	IV_FEATURE_PROFILES	FeatureProfilesUpLayer + FeatureProfilesDownLayer (legacy support)
FeatureProfilesUpLayer	String	IV_FEATURE_PROFILES	
FeatureProfilesDownLayer	String	IV_FEATURE_PROFILES_DOWN	
AltRepFrontLayer	String	IV_ALTREP_FRONT	
AltRepBackLayer	String	IV_ALTREP_BACK	
UnconsumedSketchesLayer	String	IV_UNCONSUMED_SKETCHES	
UnconsumedSketchConstructionLayer	String	IV_UNCONSUMED_SKETCH_CONSTRUCTION	
TangentRollLinesLayer	String	IV_ROLL_TANGENT	
RollLinesLayer	String	IV_ROLL	
***Color	String		*** indicates name of layer from the argument column. RGB values separated by ;. Example: TangentLayerColor=255;0;0
***LineType	Long		*** indicates name of layer from the argument column. Long value from LineTypeEnum. Example: TangentLayerLineType=37644
***LineWeight	Double		*** indicates name of layer from the argument column. Value in centimeters. Example: TangentLayerLineWeight=.1016
CustomizeFilename	String		
AcadVersion	String		2018, 2013, 2010, 2007, 2004, 2000, or R12 (for DXF only)
SimplifySplines	Boolean	True	Enable spline replacement(by linear segments or arcs).
SplineTolerance	Double	0.01	Chord tolerance for spline replacement
AdvancedLegacyExport	Boolean	True	
MergeProfilesIntoPolyline	Boolean	False	Build a polyline of the exterior profiles
RebaseGeometry	Boolean	False	Move geometry to 1st quadrant
InvisibleLayers	String		List of layer names to make invisible, separated by ;
TrimCenterlinesAtContour	Boolean	False	Trim the centerlines at contour.
SimplifyAsTangentArcs	Boolean	False	True: Replace splines by tangent arcs. False: Replace splines by line segments. The SimplifySplines should be specified to True otherwise this option is ignored.

Code Samples

- VBA

- iLogic

The following sample demonstrates creating an R12 DXF file that will have a layer called Outer where the the curves for the outer shape will be created.

Copy Code

```
Public Sub WriteSheetMetalDXF()
    ' Get the active document.  This assumes it is a part document.
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    ' Get the DataIO object.
    Dim oDataIO As DataIO
    Set oDataIO = oDoc.ComponentDefinition.DataIO

    ' Build the string that defines the format of the DXF file.
    Dim sOut As String
    sOut = "FLAT PATTERN DXF?AcadVersion=R12&OuterProfileLayer=Outer&TrimCenterlinesAtContour=True&InvisibleLayers=IV_UNCONSUMED_SKETCH

    ' Create the DXF file.
    oDataIO.WriteDataToFile sOut, "C:\temp\flat2.dxf"
End Sub
```

The following sample demonstrates creating an R12 DXF file that will have a layer called Outer where the the curves for the outer shape will be created.

Copy Code

```
    ' Get the active document.  This assumes it is a part document.
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    ' Get the DataIO object.
    Dim oDataIO As DataIO
    oDataIO = oDoc.ComponentDefinition.DataIO

    ' Build the string that defines the format of the DXF file.
    Dim sOut As String
    sOut = "FLAT PATTERN DXF?AcadVersion=R12&OuterProfileLayer=Outer&TrimCenterlinesAtContour=True&InvisibleLayers=IV_UNCONSUMED_SKETCH

    ' Create the DXF file.
    oDataIO.WriteDataToFile(sOut, "C:\temp\flat2.dxf")
```

Publish FlatPattern to SAT

Description

This sample demonstrates how to save a FlatPattern file using the SAT translator add-in.

Code Samples

- VBA
- iLogic

Open a sheet metal file before running this sample code.

Copy Code

```
Sub ExportFlatPatternToSAT()
    'Set a reference to the active document (the document to be published).
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    Set oFlatPattern = oCompDef.FlatPattern

    Dim oSATTranslator As TranslatorAddIn
    Set oSATTranslator = ThisApplication.ApplicationAddIns.ItemById("{89162634-02B6-11D5-8E80-0010B541CD80}")

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext

    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    If oSATTranslator.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then

        'oOptions.Value("IncludeSketches") = True
        'oOptions.Value("ExportBodyNames") = True
        oContext.Type = IOMechanismEnum.kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = oDoc.FullFileName & ".sat"

    Call oSATTranslator.SaveCopyAs(oFlatPattern, oContext, oOptions, oData)
    End If
End Sub
```

Open a sheet metal file before running this sample code.

Copy Code

```
    'Set a reference to the active document (the document to be published).
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    oFlatPattern = oCompDef.FlatPattern

    Dim oSATTranslator As TranslatorAddIn
    oSATTranslator = ThisApplication.ApplicationAddIns.ItemById("{89162634-02B6-11D5-8E80-0010B541CD80}")

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext

    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    If oSATTranslator.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then

        'oOptions.Value("IncludeSketches") = True
        'oOptions.Value("ExportBodyNames") = True
        oContext.Type = IOMechanismEnum.kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = oDoc.FullFileName & ".sat"

        Call oSATTranslator.SaveCopyAs(oFlatPattern, oContext, oOptions, oData)
    End If
```

Publish FlatPattern to STEP

Description

This sample demonstrates how to save a FlatPattern file using the STEP translator add-in.

Code Samples

- VBA
- iLogic

This sample demonstrates how to save a FlatPattern file using the STEP translator add-in.

Copy Code

```
Sub ExportFlatPatternToSTEP()
    'Set a reference to the active document (the document to be published).
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oDoc.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    Set oFlatPattern = oCompDef.FlatPattern

    Dim oSTEPTranslator As TranslatorAddIn
    Set oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext

    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    If oSTEPTranslator.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3
        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = IOMechanismEnum.kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = oDoc.FullFileName & ".stp"
    Call oSTEPTranslator.SaveCopyAs(oFlatPattern, oContext, oOptions, oData)
    End If
End Sub
```

This sample demonstrates how to save a FlatPattern file using the STEP translator add-in.

Copy Code

```
    'Set a reference to the active document (the document to be published).
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oDoc.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    oFlatPattern = oCompDef.FlatPattern

    Dim oSTEPTranslator As TranslatorAddIn
    oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext

    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    If oSTEPTranslator.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3
        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = IOMechanismEnum.kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = oDoc.FullFileName & ".stp"
    Call oSTEPTranslator.SaveCopyAs(oFlatPattern, oContext, oOptions, oData)
    End If
```

Export to 3D PDF

Description

The 3D PDF converter is to translate Inventor model to 3D PDF format, but not like the other translators this function is implemented in an ApplicationAddin, so it is a bit different to be used via API. Below sample demonstrates how to use the ApplicationAddin.Automation to get the function to export Inventor model to 3D PDF.

Code Samples

- VBA
- iLogic
- C#
- C++

Open a part document, and new a design view named "View1", and run below VBA code:

Copy Code

```
Public Sub PublishTo3DPDF()
    ' Get the 3D PDF Add-In.
    Dim oPDFAddIn As ApplicationAddIn
    Dim oAddin As ApplicationAddIn
    For Each oAddin In ThisApplication.ApplicationAddIns
        If oAddin.ClassIdString = "{3EE52B28-D6E0-4EA4-8AA6-C2A266DEBB88}" Then
            Set oPDFAddIn = oAddin
            Exit For
        End If
    Next

    If oPDFAddIn Is Nothing Then
        MsgBox "Inventor 3D PDF Addin not loaded."
        Exit Sub
    End If

    Dim oPDFConvertor3D
    Set oPDFConvertor3D = oPDFAddIn.Automation

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    ' Create a NameValueMap object as Options
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Options
    oOptions.Value("FileOutputLocation") = "c:\temp\test.pdf"
    oOptions.Value("ExportAnnotations") = 1
    oOptions.Value("ExportWokFeatures") = 1
    oOptions.Value("GenerateAndAttachSTEPFile") = True
    oOptions.Value("VisualizationQuality") = kHigh

    ' Set the properties to export
    Dim sProps(0) As String
    sProps(0) = "{F29F85E0-4FF9-1068-AB91-08002B27B3D9}:Title"  ' Title
```

```
    oOptions.Value("ExportAllProperties") = False
    oOptions.Value("ExportProperties") = sProps

    ' Set the design views to export
    Dim sDesignViews(1) As String
    sDesignViews(0) = "Master"
    sDesignViews(1) = "View1"

    oOptions.Value("ExportDesignViewRepresentations") = sDesignViews

    'Publish document.
    Call oPDFConvertor3D.Publish(oDocument, oOptions)
End Sub
```

Open a part document, and new a design view named "View1", and run below VBA code:

Copy Code

```
    ' Get the 3D PDF Add-In.
    Dim oPDFAddIn As ApplicationAddIn
    Dim oAddin As ApplicationAddIn
    For Each oAddin In ThisApplication.ApplicationAddIns
        If oAddin.ClassIdString = "{3EE52B28-D6E0-4EA4-8AA6-C2A266DEBB88}" Then
            oPDFAddIn = oAddin
            Exit For
        End If
    Next

    If oPDFAddIn Is Nothing Then
        MsgBox("Inventor 3D PDF Addin not loaded.")
        Exit Sub
    End If

    Dim oPDFConvertor3D
    oPDFConvertor3D = oPDFAddIn.Automation

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    ' Create a NameValueMap object as Options
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Options
    oOptions.Value("FileOutputLocation") = "c:\temp\test.pdf"
    oOptions.Value("ExportAnnotations") = 1
    oOptions.Value("ExportWokFeatures") = 1
    oOptions.Value("GenerateAndAttachSTEPFile") = True
    oOptions.Value("VisualizationQuality") = kHigh

    ' Set the properties to export
    Dim sProps(0) As String
    sProps(0) = "{F29F85E0-4FF9-1068-AB91-08002B27B3D9}:Title"  ' Title

    oOptions.Value("ExportAllProperties") = False
    oOptions.Value("ExportProperties") = sProps

    ' Set the design views to export
    Dim sDesignViews(1) As String
    sDesignViews(0) = "Master"
    sDesignViews(1) = "View1"

    oOptions.Value("ExportDesignViewRepresentations") = sDesignViews

    'Publish document.
    Call oPDFConvertor3D.Publish(oDocument, oOptions)
```

Open a part document, and new a design view named "View1", and copy below C# code to a project to run it.

Copy Code

```
public void Export3DPdf()
        {
            Inventor.Application InvApp = mApp;

            Inventor.ApplicationAddIn  PDFAddin = null ;

            foreach (ApplicationAddIn appAddin in mApp.ApplicationAddIns )
            {
                if (appAddin.ClassIdString == "{3EE52B28-D6E0-4EA4-8AA6-C2A266DEBB88}")
                {
                    PDFAddin =  appAddin;
                    break;
                }
            }

            dynamic pdfConvertor3d = PDFAddin.Automation;

            Document oDoc = mApp.ActiveDocument;
            // Create a NameValueMap object
            Inventor.NameValueMap oOptions = InvApp.TransientObjects.CreateNameValueMap();

            oOptions.Value["FileOutputLocation"] = @"c:\temp\3DPDF.pdf";
            oOptions.Value["ExportAllProperties"] = true;
            oOptions.Value["GenerateAndAttachSTEPFile"] = true ;
            oOptions.Value["ExportTemplate"] = @"C:\Users\Public\Documents\Autodesk\Inventor 2019\Templates\Sample Part Template.pdf";
            oOptions.Value["VisualizationQuality"] = AccuracyEnum.kHigh;

            //string[] sProps = new string[1];
            //sProps[0] = "{F29F85E0-4FF9-1068-AB91-08002B27B3D9}:Title";
            //oOptions.Value["ExportAllProperties"] = false;
            //oOptions.Value["ExportProperties"] = sProps;

            String[] sDesignViews = new string[] { "Master", "View1" };
            oOptions.Value["ExportDesignViewRepresentations"] = sDesignViews;

            pdfConvertor3d.Publish(oDoc, oOptions);
```

```
}
```

Copy the C++ code to your project, and change the hardcoded file names for the sFileName in ExportToPDF3D to set the document path in your machine, and update the sTemplatePath to specify a proper template according to the doucment type. Then call the ExportToPDF3D to export the document to 3D PDF.

Copy Code

```
static HRESULT ExportToPDF3D();
DesignViewRepresentationsPtr GetDesignViewRepresentations(Document* pDocument);
HRESULT GetDesignViews(DesignViewRepresentationsPtr pDVRepresentations, CComSafeArray<BSTR>& sDVs);
HRESULT CreateStepFileOption(NameValueMap *pOptionsStep);


static HRESULT ExportToPDF3D()
{
	HRESULT hr = NOERROR;

	TCHAR Str[_MAX_PATH];

	CLSID InvAppClsid;
	hr = CLSIDFromProgID (L"Inventor.Application", &InvAppClsid);
	if (FAILED(hr)) return hr;

	// Either create a new instance of the application or latch on to the currently active one.
	_tprintf_s (_T("Would you like to create a new Inventor application? (y/n) _: "));
	_tscanf_s (_T("%ls"), Str, _MAX_PATH);

	CComPtr<IUnknown> pInvAppUnk;
	if (toupper (Str[0]) == _T('Y'))
	{
		hr = CoCreateInstance (InvAppClsid, NULL, CLSCTX_LOCAL_SERVER, __uuidof(IUnknown), (void **) &pInvAppUnk);
		if (FAILED (hr))
			_tprintf_s (_T("*** Failed to create a new Inventor application ***\n"));
	}
	else
	{
		hr = ::GetActiveObject (InvAppClsid, NULL, &pInvAppUnk);
		if (FAILED (hr))
			_tprintf_s (_T("*** Could not get hold of an active Inventor application ***\n"));
	}
	if (FAILED(hr)) return hr;

	CComPtr<Application> m_pApplication;//m_pApplication;
	hr = pInvAppUnk->QueryInterface (__uuidof(Application), (void **) &m_pApplication);
	if (FAILED(hr)) return hr;

	hr = m_pApplication->put_Visible(-1);
	if (FAILED(hr)) return hr;

	TransientObjectsPtr pTransObj;
	hr = m_pApplication->get_TransientObjects(&pTransObj);
	if (FAILED(hr))		return hr;

	CComBSTR sFileName = "C:\\Temp\\Assembly1.iam";

	CComPtr<Documents> pDocs;
	hr = m_pApplication->get_Documents(&pDocs);
	if (FAILED(hr)) return hr;

	DocumentPtr pDoc = nullptr;
	hr = pDocs->Open(sFileName, VARIANT_TRUE, &pDoc);
	if (FAILED(hr)) return hr;

	//get NameValueMap
	CComPtr<NameValueMap> pOptionsPDF = nullptr;
	hr = pTransObj->CreateNameValueMap(&pOptionsPDF);
	if (FAILED(hr))			return hr;

	// Set the option "FileOutputLocation" to NameValueMap
	CComBSTR sOutputFilePath = "C:\\Users\\Temp\\Assembly1.pdf";
	hr = pOptionsPDF->MethodAdd(_T("FileOutputLocation"), CComVariant(sOutputFilePath));
	if (FAILED(hr))			return hr;


	// Set the option "ExportTemplate" to NameValueMap
	CComBSTR sTemplatePath = "C:\\Users\\Public\\Documents\\Autodesk\\Inventor 2019\\Templates\\Sample Assembly Template.pdf";
	hr = pOptionsPDF->MethodAdd(_T("ExportTemplate"), CComVariant(sTemplatePath));
	if (FAILED(hr))		return hr;

	// Set the option "VisualizationQuality" to NameValueMap
	AccuracyEnum eVisualizationQuality = kHigh; // kLow  kMedium

	hr = pOptionsPDF->MethodAdd(_T("VisualizationQuality"), CComVariant(eVisualizationQuality));
	if (FAILED(hr))		return hr;

	// Set the option "LimitToEntitiesInDVRs" to NameValueMap
	bool  m_bPublishAllEntities = false;
	hr = pOptionsPDF->MethodAdd(_T("LimitToEntitiesInDVRs"), CComVariant(m_bPublishAllEntities));
	if (FAILED(hr))		return hr;

	// Set the option "ExportAllProperties" to NameValueMap
	bool bExportAllProperties = true;
	hr = pOptionsPDF->MethodAdd(_T("ExportAllProperties"), CComVariant(bExportAllProperties));
	if (FAILED(hr))		return hr;

	if (!bExportAllProperties)
	{
		// Set the option "ExportProperties" to NameValueMap
		CComSafeArray<BSTR> saProperties;
		saProperties.Add(CComBSTR("{F29F85E0-4FF9-1068-AB91-08002B27B3D9}:Title"));
		saProperties.Add(CComBSTR("{D5CDD502-2E9C-101B-9397-08002B2CF9AE}:Company"));
		saProperties.Add(CComBSTR("{32853F0F-3444-11D1-9E93-0060B03C1CA6}:Part Number"));

		hr = pOptionsPDF->MethodAdd(_T("ExportProperties"), CComVariant(saProperties));
		if (FAILED(hr))			return hr;
	}
```

```
	// Set the option "ExportDesignViewRepresentations" to NameValueMap
	DesignViewRepresentationsPtr spDVReps = GetDesignViewRepresentations(pDoc);
	CComSafeArray<BSTR> sDesignViews;

	hr = GetDesignViews(spDVReps, sDesignViews);
	if (FAILED(hr))		return hr;

	if (sDesignViews != nullptr)
	{
		hr = pOptionsPDF->MethodAdd(_T("ExportDesignViewRepresentations"), CComVariant(sDesignViews));
		if (FAILED(hr))			return hr;
	}

	// Set the option "GenerateAndAttachSTEPFile" to NameValueMap
	bool bAttachSTEPFile = true;
	hr = pOptionsPDF->MethodAdd(_T("GenerateAndAttachSTEPFile"), CComVariant(bAttachSTEPFile));
	if (FAILED(hr))		return hr;

	if (bAttachSTEPFile)
	{
		//get NameValueMap
		CComPtr<NameValueMap> pOptionsStep = nullptr;
		hr = pTransObj->CreateNameValueMap(&pOptionsStep);
		if (FAILED(hr))			return hr;

		hr = CreateStepFileOption(pOptionsStep);
		if (FAILED(hr))			return hr;

		// Set the option "STEPFileOptions" to NameValueMap
		hr = pOptionsPDF->MethodAdd(_T("STEPFileOptions"), CComVariant(pOptionsStep));
		if (FAILED(hr))			return hr;
	}

	// Set the option "AttachedFiles" to NameValueMap
	CComSafeArray<BSTR> aAttachFiles;
	hr = aAttachFiles.Add(CComBSTR("C:\\temp\\log.pdf"));
	if (FAILED(hr))			return hr;

	hr = pOptionsPDF->MethodAdd(_T("AttachedFiles"), CComVariant(aAttachFiles));
	if (FAILED(hr))			return hr;

	CComPtr<ApplicationAddIns> pInvAppAddIns = nullptr;
	hr = m_pApplication->get_ApplicationAddIns(&pInvAppAddIns);
	if (FAILED(hr))		return hr;

	// Client id of 3D PDf addin
	CComBSTR clsidAddin = CComBSTR(_T("{3EE52B28-D6E0-4EA4-8AA6-C2A266DEBB88}"));

	CComPtr<ApplicationAddIn> pAddin = nullptr;
	hr = pInvAppAddIns->get_ItemById(clsidAddin, &pAddin);

	if (pAddin)
	{
		CComPtr<IDispatch> pAddinAutomaton;
		pAddin->get_Automation(&pAddinAutomaton);

		if (pAddinAutomaton)
		{
			DISPID dispId;
			OLECHAR* name(OLESTR("Publish"));
			hr = pAddinAutomaton->GetIDsOfNames(IID_NULL, &name, 1, LOCALE_SYSTEM_DEFAULT, &dispId);

			if (SUCCEEDED(hr))
			{
				CComVariant varArgs[2];
				varArgs[1] = pDoc.GetInterfacePtr();
				varArgs[0] = pOptionsPDF;
				DISPPARAMS params = { &varArgs[0], NULL, 2, 0 };
				CComVariant vRes;

				hr = pAddinAutomaton->Invoke(dispId, IID_NULL, LOCALE_SYSTEM_DEFAULT, DISPATCH_METHOD, ¶ms, &vRes, NUL
			}
		}
	}

	return hr;
}

DesignViewRepresentationsPtr GetDesignViewRepresentations(Document* pDocument)
{
	if (pDocument == nullptr)
		return nullptr;

	HRESULT hr = S_OK;
	RepresentationsManagerPtr pRepsMgr;

	AssemblyDocumentPtr spAsmDoc = pDocument;
	if (spAsmDoc != nullptr)
	{
		AssemblyComponentDefinitionPtr pAsmDef;
		hr = spAsmDoc->get_ComponentDefinition(&pAsmDef);
		if (FAILED(hr))
			return nullptr;

		hr = pAsmDef->get_RepresentationsManager(&pRepsMgr);
		if (FAILED(hr))
			return nullptr;
	}
	else
	{
		PartDocumentPtr pPartDoc = pDocument;
		if (pPartDoc != nullptr)
		{
			PartComponentDefinitionPtr pPartDef;
			hr = pPartDoc->get_ComponentDefinition(&pPartDef);
			if (FAILED(hr))
```

```
                                return nullptr;

                        hr = pPartDef->get_RepresentationsManager(&pRepsMgr);
                        if (FAILED(hr))
                                return nullptr;
                }
        }

        if (pRepsMgr != nullptr)
        {
                DesignViewRepresentationsPtr pDVRepresentations;
                hr = pRepsMgr->get_DesignViewRepresentations(&pDVRepresentations);
                if (FAILED(hr))
                        return nullptr;

                return pDVRepresentations;
        }

        return nullptr;
}

HRESULT GetDesignViews(DesignViewRepresentationsPtr pDVRepresentations, CComSafeArray<BSTR>& sDVs)
{
        HRESULT hr = S_OK;



        long lCountDV = 0;
        hr = pDVRepresentations->get_Count(&lCountDV);
        if (FAILED(hr))
                return hr;

        for (int i = 0; i < lCountDV; i++)
        {
                DesignViewRepresentationPtr pDV = nullptr;
                CComVariant varIndex(i + 1);
                hr = pDVRepresentations->get_Item(varIndex, &pDV);
                if (FAILED(hr))
                        continue;

                DesignViewTypeEnum viewType;
                pDV->get_DesignViewType(&viewType);
                if (viewType == kTransientDesignViewType)
                        continue;

                if (VARIANT_TRUE) {
                        VARIANT_BOOL bPublishDV;
                        hr = pDV->get_Publish(&bPublishDV);

                        if (FAILED(hr) || bPublishDV == VARIANT_FALSE)
                                continue;
                }

                CComBSTR bstrNameDV;
                pDV->get_Name(&bstrNameDV);
                sDVs.Add(CComBSTR(bstrNameDV));
        }

        return S_OK;
}

HRESULT CreateStepFileOption(NameValueMap *pOptionsStep)
{
        HRESULT hr = S_OK;

        // check input map pointer
        if (pOptionsStep == nullptr)
                return S_FALSE;

        // Set the option "ApplicationProtocolType" to NameValueMap
        // Valid values: eAP203 = 2, eAP214IS = 4 and eAP242 = 5
        int iApplicationProtocolType = 5;
        hr = pOptionsStep->MethodAdd(_T("ApplicationProtocolType"), CComVariant(iApplicationProtocolType));
        if (FAILED(hr))         return hr;

        // Set the option "Author" to NameValueMap
        CComBSTR sAuthor = "";
        hr = pOptionsStep->MethodAdd(_T("Author"), CComVariant(sAuthor));
        if (FAILED(hr))         return hr;

        // Set the option "Authorization" to NameValueMap
        CComBSTR sAuthorization = "";
        hr = pOptionsStep->MethodAdd(_T("Authorization"), CComVariant(sAuthorization));
        if (FAILED(hr))         return hr;

        // Set the option "Description" to NameValueMap
        CComBSTR sDescription = "";
        hr = pOptionsStep->MethodAdd(_T("Description"), CComVariant(sDescription));
        if (FAILED(hr))
                return hr;

        // Set the option "ExportFitTolerance" to NameValueMap
        double dTolerance = 0.001;
        hr = pOptionsStep->MethodAdd(_T("ExportFitTolerance"), CComVariant(dTolerance));
        if (FAILED(hr))         return hr;

        // Set the option "Organization" to NameValueMap
        CComBSTR sOrganization = "";
        hr = pOptionsStep->MethodAdd(_T("Organization"), CComVariant(sOrganization));
        if (FAILED(hr))         return hr;

        // Set the option "IncludeSketches" to NameValueMap
        bool bExportSketches = true;
        hr = pOptionsStep->MethodAdd(_T("IncludeSketches"), CComVariant(bExportSketches));
        if (FAILED(hr))         return hr;
```

```
        return hr;
}
```

Export to USDz

Description

This sample demonstrates how to export a part or assembly document to USDz format.

Code Samples

- VBA
- iLogic

This sample demonstrates how to export a part or assembly document to USDz format.

Copy Code

```
Sub SaveToUSDz()
  Dim oAddin As TranslatorAddIn
  Set oAddin = ThisApplication.ApplicationAddIns.ItemById("{2F08D88C-E86A-490E-9059-DD97A44020AC}")

  Dim oContext As TranslationContext
  Set oContext = ThisApplication.TransientObjects.CreateTranslationContext()
  oContext.Type = kFileBrowseIOMechanism

  Dim oOptions As NameValueMap
  Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap()

  Dim oData As DataMedium
  Set oData = ThisApplication.TransientObjects.CreateDataMedium()

  Dim outputFolderPath As String
  outputFolderPath = "C:\Temp\"
  oData.FileName = outputFolderPath + "USDz_output.usdz"

  Dim oDoc As Document
  Set oDoc = ThisApplication.ActiveDocument
  If oDoc.DocumentType = kAssemblyDocumentObject Or oDoc.DocumentType = kPartDocumentObject Then
    Call oAddin.SaveCopyAs(oDoc, oContext, oOptions, oData)
  End If
End Sub
```

This sample demonstrates how to export a part or assembly document to USDz format.

Copy Code

```
    Dim oAddin As TranslatorAddIn
  oAddin = ThisApplication.ApplicationAddIns.ItemById("{2F08D88C-E86A-490E-9059-DD97A44020AC}")

  Dim oContext As TranslationContext
  oContext = ThisApplication.TransientObjects.CreateTranslationContext()
  oContext.Type = kFileBrowseIOMechanism

  Dim oOptions As NameValueMap
  oOptions = ThisApplication.TransientObjects.CreateNameValueMap()

  Dim oData As DataMedium
  oData = ThisApplication.TransientObjects.CreateDataMedium()

  Dim outputFolderPath As String
  outputFolderPath = "C:\Temp\"
  oData.FileName = outputFolderPath + "USDz_output.usdz"

  Dim oDoc As Document
  oDoc = ThisApplication.ActiveDocument
  If oDoc.DocumentType = kAssemblyDocumentObject Or oDoc.DocumentType = kPartDocumentObject Then
    Call oAddin.SaveCopyAs(oDoc, oContext, oOptions, oData)
  End If
```

Publish FlatPattern to DXF

Description

This sample demonstrates how to save a FlatPattern file using the DXF translator add-in.

Code Samples

- VBA
- iLogic

This sample demonstrates how to save a FlatPattern file using the DXF translator add-in.

Copy Code

```
Public Sub PublishFlatPatternToDXF()
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    Set DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")
```

```
    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    Set oCompDef = oDocument.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    Set oFlatPattern = oCompDef.FlatPattern

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then
        ' You can prepare a configutation .ini file from UI.
        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oFlatPattern, oContext, oOptions, oDataMedium)
End Sub
```

This sample demonstrates how to save a FlatPattern file using the DXF translator add-in.

Copy Code

```
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oCompDef As SheetMetalComponentDefinition
    oCompDef = oDocument.ComponentDefinition

    If oCompDef.HasFlatPattern = False Then
        oCompDef.Unfold
    End If

    Dim oFlatPattern As FlatPattern
    oFlatPattern = oCompDef.FlatPattern

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oFlatPattern, oContext, oOptions) Then
        ' You can prepare a configutation .ini file from UI.
        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oFlatPattern, oContext, oOptions, oDataMedium)
```

Save as DWF Translator Sample

Description

This sample demonstrates how to save a DWF file using the DWF translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishDWF()
    ' Get the DWF translator Add-In.
    Dim DWFAddIn As TranslatorAddIn
    Set DWFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        oOptions.Value("Launch_Viewer") = 1

        ' Other options...
        'oOptions.Value("Publish_All_Component_Props") = 1
        'oOptions.Value("Publish_All_Physical_Props") = 1
        'oOptions.Value("Password") = 0

        If Typeof oDocument Is DrawingDocument Then

            ' Drawing options
            oOptions.Value("Publish_Mode") = kCustomDWFPublish
            oOptions.Value("Publish_All_Sheets") = 0

            ' The specified sheets will be ignored if
            ' the option "Publish_All_Sheets" is True (1)
            Dim oSheets As NameValueMap
            Set oSheets = ThisApplication.TransientObjects.CreateNameValueMap

            ' Publish the first sheet AND its 3D model
            Dim oSheet1Options As NameValueMap
            Set oSheet1Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet1Options.Add "Name", "Sheet:1"
            oSheet1Options.Add "3DModel", True
            oSheets.Value("Sheet1") = oSheet1Options

            ' Publish the third sheet but NOT its 3D model
            Dim oSheet3Options As NameValueMap
            Set oSheet3Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet3Options.Add "Name", "Sheet:3"
            oSheet3Options.Add "3DModel", False

            oSheets.Value("Sheet2") = oSheet3Options

            'Set the sheet options object in the oOptions NameValueMap
            oOptions.Value("Sheets") = oSheets
        End If

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.dwf"

    'Publish document.
    Call DWFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the DWF translator Add-In.
    Dim DWFAddIn As TranslatorAddIn
    DWFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then
```

```
        oOptions.Value("Launch_Viewer") = 1

        ' Other options...
        'oOptions.Value("Publish_All_Component_Props") = 1
        'oOptions.Value("Publish_All_Physical_Props") = 1
        'oOptions.Value("Password") = 0

        If Typeof oDocument Is DrawingDocument Then

            ' Drawing options
            oOptions.Value("Publish_Mode") = kCustomDWFPublish
            oOptions.Value("Publish_All_Sheets") = 0

            ' The specified sheets will be ignored if
            ' the option "Publish_All_Sheets" is True (1)
            Dim oSheets As NameValueMap
            oSheets = ThisApplication.TransientObjects.CreateNameValueMap

            ' Publish the first sheet AND its 3D model
            Dim oSheet1Options As NameValueMap
            oSheet1Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet1Options.Add("Name", "Sheet:1")
            oSheet1Options.Add("3DModel", True)
            oSheets.Value("Sheet1") = oSheet1Options

            ' Publish the third sheet but NOT its 3D model
            Dim oSheet3Options As NameValueMap
            oSheet3Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet3Options.Add("Name", "Sheet:3")
            oSheet3Options.Add("3DModel", False)

            oSheets.Value("Sheet2") = oSheet3Options

            'Set the sheet options object in the oOptions NameValueMap
            oOptions.Value("Sheets") = oSheets
        End If

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.dwf"

    'Publish document.
    Call DWFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Save as DWG Translator Sample

Description

This sample demonstrates how to save a DWG file using the DWG translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishDWG()
    ' Get the DWG translator Add-In.
    Dim DWGAddIn As TranslatorAddIn
    Set DWGAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWGAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DWGOut.ini"
        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dwgout.dwg"

    'Publish document.
    Call DWGAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the DWG translator Add-In.
    Dim DWGAddIn As TranslatorAddIn
    DWGAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWGAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DWGOut.ini"
        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dwgout.dwg"

    'Publish document.
    Call DWGAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Save as DXF Translator Sample

Description

This sample demonstrates how to save a DXF file using the DXF translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishDXF()
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    Set DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then
        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument
```

```
    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then
        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Save as IGES Translator Sample

Description

This sample demonstrates how to save a IGES file using the IGES translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ExportToIGES()
    ' Get the IGES translator Add-In.
    Dim oIGESTranslator As TranslatorAddIn
    Set oIGESTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F44-0C01-11D5-8E83-0010B541CD80}")

    If oIGESTranslator Is Nothing Then
        MsgBox "Could not access IGES translator."
        Exit Sub
    End If

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oIGESTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set geometry type for wireframe.
        ' 0 = Surfaces, 1 = Solids, 2 = Wireframe
        oOptions.Value("GeometryType") = 1

        ' To set other translator values:
        ' oOptions.Value("SolidFaceType") = n
        ' 0 = NURBS, 1 = Analytic

        ' oOptions.Value("SurfaceType") = n
        ' 0 = 143(Bounded), 1 = 144(Trimmed)

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\Temptest.igs"

        Call oIGESTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
End Sub
```

Copy Code

```
    ' Get the IGES translator Add-In.
    Dim oIGESTranslator As TranslatorAddIn
    oIGESTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F44-0C01-11D5-8E83-0010B541CD80}")

    If oIGESTranslator Is Nothing Then
        MsgBox("Could not access IGES translator.")
        Exit Sub
    End If

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oIGESTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set geometry type for wireframe.
        ' 0 = Surfaces, 1 = Solids, 2 = Wireframe
        oOptions.Value("GeometryType") = 1
```

```
        ' To set other translator values:
        ' oOptions.Value("SolidFaceType") = n
        ' 0 = NURBS, 1 = Analytic

        ' oOptions.Value("SurfaceType") = n
        ' 0 = 143(Bounded), 1 = 144(Trimmed)

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\Temptest.igs"

        Call oIGESTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
```

Save as PDF Translator Sample

Description

This sample demonstrates how to save a PDF file using the PDF translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishPDF()
    ' Get the PDF translator Add-In.
    Dim PDFAddIn As TranslatorAddIn
    Set PDFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD96-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If PDFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        ' Options for drawings...

        oOptions.Value("All_Color_AS_Black") = 0

        'oOptions.Value("Remove_Line_Weights") = 0
        'oOptions.Value("Vector_Resolution") = 400
        'oOptions.Value("Sheet_Range") = kPrintAllSheets
        'oOptions.Value("Custom_Begin_Sheet") = 2
        'oOptions.Value("Custom_End_Sheet") = 4
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.pdf"

    'Publish document.
    Call PDFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the PDF translator Add-In.
    Dim PDFAddIn As TranslatorAddIn
    PDFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD96-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If PDFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        ' Options for drawings...
```

```
        oOptions.Value("All_Color_AS_Black") = 0

        'oOptions.Value("Remove_Line_Weights") = 0
        'oOptions.Value("Vector_Resolution") = 400
        'oOptions.Value("Sheet_Range") = kPrintAllSheets
        'oOptions.Value("Custom_Begin_Sheet") = 2
        'oOptions.Value("Custom_End_Sheet") = 4
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.pdf"

    'Publish document.
    Call PDFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Save as STEP Translator Sample

Description

This sample demonstrates how to save a STEP file using the STEP translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ExportToSTEP()
    ' Get the STEP translator Add-In.
    Dim oSTEPTranslator As TranslatorAddIn
    Set oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    If oSTEPTranslator Is Nothing Then
        MsgBox "Could not access STEP translator."
        Exit Sub
    End If

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oSTEPTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3

        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\temp\test.stp"

        Call oSTEPTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
End Sub
```

Copy Code

```
    ' Get the STEP translator Add-In.
    Dim oSTEPTranslator As TranslatorAddIn
    oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    If oSTEPTranslator Is Nothing Then
        MsgBox("Could not access STEP translator.")
        Exit Sub
    End If

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oSTEPTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3

        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\temp\test.stp"
```

```
        Call oSTEPTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
```

Export to DWF

Description

This sample demonstrates publishing of Inventor files in DWF format.

Code Samples

- VBA
- iLogic

To run this sample, the document to be published must be active. The DWF file is created as C:\temptest.DWF (this can be changed in the code below).

Copy Code

```
Public Sub PublishDWF()
    ' Get the DWF translator Add-In.
    Dim DWFAddIn As TranslatorAddIn
    Set DWFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        oOptions.Value("Launch_Viewer") = 1

        ' Other options...
        'oOptions.Value("Publish_All_Component_Props") = 1
        'oOptions.Value("Publish_All_Physical_Props") = 1
        'oOptions.Value("Password") = 0

        If Typeof oDocument Is DrawingDocument Then

            ' Drawing options
            oOptions.Value("Publish_Mode") = kCustomDWFPublish
            oOptions.Value("Publish_All_Sheets") = 0

            ' The specified sheets will be ignored if
            ' the option "Publish_All_Sheets" is True (1)
            Dim oSheets As NameValueMap
            Set oSheets = ThisApplication.TransientObjects.CreateNameValueMap

            ' Publish the first sheet AND its 3D model
            Dim oSheet1Options As NameValueMap
            Set oSheet1Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet1Options.Add "Name", "Sheet:1"
            oSheet1Options.Add "3DModel", True
            oSheets.Value("Sheet1") = oSheet1Options

            ' Publish the third sheet but NOT its 3D model
            Dim oSheet3Options As NameValueMap
            Set oSheet3Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet3Options.Add "Name", "Sheet3:3"
            oSheet3Options.Add "3DModel", False

            oSheets.Value("Sheet2") = oSheet3Options

            'Set the sheet options object in the oOptions NameValueMap
            oOptions.Value("Sheets") = oSheets
        End If

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.dwf"

    'Publish document.
    Call DWFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

To run this sample, the document to be published must be active. The DWF file is created as C:\temptest.DWF (this can be changed in the code below).

Copy Code

```
    ' Get the DWF translator Add-In.
    Dim DWFAddIn As TranslatorAddIn
    DWFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}")
```

```
    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        oOptions.Value("Launch_Viewer") = 1

        ' Other options...
        'oOptions.Value("Publish_All_Component_Props") = 1
        'oOptions.Value("Publish_All_Physical_Props") = 1
        'oOptions.Value("Password") = 0

        If Typeof oDocument Is DrawingDocument Then

            ' Drawing options
            oOptions.Value("Publish_Mode") = kCustomDWFPublish
            oOptions.Value("Publish_All_Sheets") = 0

            ' The specified sheets will be ignored if
            ' the option "Publish_All_Sheets" is True (1)
            Dim oSheets As NameValueMap
            oSheets = ThisApplication.TransientObjects.CreateNameValueMap

            ' Publish the first sheet AND its 3D model
            Dim oSheet1Options As NameValueMap
            oSheet1Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet1Options.Add("Name", "Sheet:1")
            oSheet1Options.Add("3DModel", True)
            oSheets.Value("Sheet1") = oSheet1Options

            ' Publish the third sheet but NOT its 3D model
            Dim oSheet3Options As NameValueMap
            oSheet3Options = ThisApplication.TransientObjects.CreateNameValueMap

            oSheet3Options.Add("Name", "Sheet3:3")
            oSheet3Options.Add("3DModel", False)

            oSheets.Value("Sheet2") = oSheet3Options

            'Set the sheet options object in the oOptions NameValueMap
            oOptions.Value("Sheets") = oSheets
        End If

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.dwf"

    'Publish document.
    Call DWFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Export to DWG

Description

This sample uses the DWG Translator Addin to publish to DWG.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishDWG()
    ' Get the DWG translator Add-In.
    Dim DWGAddIn As TranslatorAddIn
    Set DWGAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
```

```
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWGAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DWGOut.ini"
        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dwgout.dwg"

    'Publish document.
    Call DWGAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the DWG translator Add-In.
    Dim DWGAddIn As TranslatorAddIn
    DWGAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DWGAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DWGOut.ini"
        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dwgout.dwg"

    'Publish document.
    Call DWGAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Export to DXF

Description

This sample uses the DXF Translator Addin to publish to DXF.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PublishDXF()
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    Set DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
```

```
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

Copy Code

```
    ' Get the DXF translator Add-In.
    Dim DXFAddIn As TranslatorAddIn
    DXFAddIn = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC4-122E-11D5-8E91-0010B541CD80}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If DXFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        Dim strIniFile As String
        strIniFile = "C:\temp\DXFOut.ini"

        ' Create the name-value that specifies the ini file to use.
        oOptions.Value("Export_Acad_IniFile") = strIniFile
    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\dxfout.dxf"

    'Publish document.
    Call DXFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Export to IGES

Description

This sample demonstrates exporting of Inventor files in IGES format.

Code Samples

- VBA
- iLogic

To run this sample, the document to be exported must be active. The IGES file is created as C:\temptest.igs (this can be changed in the code below).

Copy Code

```
Public Sub ExportToIGES()
    ' Get the IGES translator Add-In.
    Dim oIGESTranslator As TranslatorAddIn
    Set oIGESTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F44-0C01-11D5-8E83-0010B541CD80}")

    If oIGESTranslator Is Nothing Then
        MsgBox "Could not access IGES translator."
        Exit Sub
    End If

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oIGESTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set geometry type for wireframe.
        ' 0 = Surfaces, 1 = Solids, 2 = Wireframe
        oOptions.Value("GeometryType") = 1

        ' To set other translator values:
        ' oOptions.Value("SolidFaceType") = n
        ' 0 = NURBS, 1 = Analytic

        ' oOptions.Value("SurfaceType") = n
        ' 0 = 143(Bounded), 1 = 144(Trimmed)

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\Temp\test.igs"

        Call oIGESTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
End Sub
```

To run this sample, the document to be exported must be active. The IGES file is created as C:\temptest.igs (this can be changed in the code below).

Copy Code

```
    ' Get the IGES translator Add-In.
    Dim oIGESTranslator As TranslatorAddIn
    oIGESTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F44-0C01-11D5-8E83-0010B541CD80}")

    If oIGESTranslator Is Nothing Then
        MsgBox("Could not access IGES translator.")
        Exit Sub
    End If

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oIGESTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set geometry type for wireframe.
        ' 0 = Surfaces, 1 = Solids, 2 = Wireframe
        oOptions.Value("GeometryType") = 1

        ' To set other translator values:
        ' oOptions.Value("SolidFaceType") = n
        ' 0 = NURBS, 1 = Analytic

        ' oOptions.Value("SurfaceType") = n
        ' 0 = 143(Bounded), 1 = 144(Trimmed)

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\Temp\test.igs"

        Call oIGESTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
```

Export to STEP

Description

This sample demonstrates exporting of Inventor files in STEP format.

Code Samples

- VBA
- iLogic

To run this sample, the document to be exported must be active. The STEP file is created as C:\temptest.stp (this can be changed in the code below).

Copy Code

```
Public Sub ExportToSTEP()
    ' Get the STEP translator Add-In.
    Dim oSTEPTranslator As TranslatorAddIn
    Set oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    If oSTEPTranslator Is Nothing Then
        MsgBox "Could not access STEP translator."
        Exit Sub
    End If

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oSTEPTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3

        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        Set oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\temp\test.stp"

        Call oSTEPTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
End Sub
```

To run this sample, the document to be exported must be active. The STEP file is created as C:\temptest.stp (this can be changed in the code below).

Copy Code

```
    ' Get the STEP translator Add-In.
    Dim oSTEPTranslator As TranslatorAddIn
    oSTEPTranslator = ThisApplication.ApplicationAddIns.ItemById("{90AF7F40-0C01-11D5-8E83-0010B541CD80}")

    If oSTEPTranslator Is Nothing Then
        MsgBox("Could not access STEP translator.")
```

```
        Exit Sub
    End If

    Dim oContext As TranslationContext
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap
    If oSTEPTranslator.HasSaveCopyAsOptions(ThisApplication.ActiveDocument, oContext, oOptions) Then
        ' Set application protocol.
        ' 2 = AP 203 - Configuration Controlled Design
        ' 3 = AP 214 - Automotive Design
        oOptions.Value("ApplicationProtocolType") = 3

        ' Other options...
        'oOptions.Value("Author") = ""
        'oOptions.Value("Authorization") = ""
        'oOptions.Value("Description") = ""
        'oOptions.Value("Organization") = ""

        oContext.Type = kFileBrowseIOMechanism

        Dim oData As DataMedium
        oData = ThisApplication.TransientObjects.CreateDataMedium
        oData.FileName = "C:\temp\test.stp"

        Call oSTEPTranslator.SaveCopyAs(ThisApplication.ActiveDocument, oContext, oOptions, oData)
    End If
```

Export to PDF

Description

This sample demonstrates exporting of Inventor files in PDF format.

Code Samples

- VBA
- iLogic

To run this sample, the document to be exported must be active.

Copy Code

```
Public Sub PublishPDF()
    ' Get the PDF translator Add-In.
    Dim PDFAddIn As TranslatorAddIn
    Set PDFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD96-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    Set oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
    Set oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If PDFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        ' Options for drawings...

        oOptions.Value("All_Color_AS_Black") = 0

        'oOptions.Value("Remove_Line_Weights") = 0
        'oOptions.Value("Vector_Resolution") = 400
        'oOptions.Value("Sheet_Range") = kPrintAllSheets
        'oOptions.Value("Custom_Begin_Sheet") = 2
        'oOptions.Value("Custom_End_Sheet") = 4

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.pdf"

    'Publish document.
    Call PDFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
End Sub
```

To run this sample, the document to be exported must be active.

Copy Code

```
    ' Get the PDF translator Add-In.
    Dim PDFAddIn As TranslatorAddIn
    PDFAddIn = ThisApplication.ApplicationAddIns.ItemById("{0AC6FD96-2F4D-42CE-8BE0-8AEA580399E4}")

    'Set a reference to the active document (the document to be published).
    Dim oDocument As Document
    oDocument = ThisApplication.ActiveDocument

    Dim oContext As TranslationContext
```

```
    oContext = ThisApplication.TransientObjects.CreateTranslationContext
    oContext.Type = kFileBrowseIOMechanism

    ' Create a NameValueMap object
    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Create a DataMedium object
    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Check whether the translator has 'SaveCopyAs' options
    If PDFAddIn.HasSaveCopyAsOptions(oDocument, oContext, oOptions) Then

        ' Options for drawings...

        oOptions.Value("All_Color_AS_Black") = 0

        'oOptions.Value("Remove_Line_Weights") = 0
        'oOptions.Value("Vector_Resolution") = 400
        'oOptions.Value("Sheet_Range") = kPrintAllSheets
        'oOptions.Value("Custom_Begin_Sheet") = 2
        'oOptions.Value("Custom_End_Sheet") = 4

    End If

    'Set the destination file name
    oDataMedium.FileName = "c:\temp\test.pdf"

    'Publish document.
    Call PDFAddIn.SaveCopyAs(oDocument, oContext, oOptions, oDataMedium)
```

Open a Catia file using the Catia Translator Sample

Description

This sample demonstrates how open an Catia file using the Catia translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub CatiaImport()
    'Set CATIA V4 translator's CLSID and the file name (*.exp and *.dlv3).
    Dim strCLSID As String
    Dim strFileName As String
    strCLSID = "{C6ACD948-E1C5-4b5b-ADEE-3ED968F8CB1A}"
    strFileName = "C:\Hello.exp"

    Dim oAddIns As ApplicationAddIns
    Set oAddIns = ThisApplication.ApplicationAddIns

    ' find the translator and get CLSID and activate it
    Dim oTransAddIn As TranslatorAddIn
    Set oTransAddIn = oAddIns.ItemById(strCLSID)
    oTransAddIn.Activate

    ' get the transient object, take it as a factory to produce other objects
    Dim transientObj As TransientObjects
    Set transientObj = ThisApplication.TransientObjects

    ' prepare the 1st parameter for Open(), the file name
    Dim file As DataMedium
    Set file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' create an empty part
    Dim activeDoc As Document
    Set activeDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' prepare the 2nd parameter for Open()
    Dim context As TranslationContext
    Set context = transientObj.CreateTranslationContext
    context.Type = kFileBrowseIOMechanism
    ' this makes sure the imported stuff be inserted in the part doc created at the top
    context.OpenIntoExisting = activeDoc

    ' prepare the 3rd parameter for Open(), the options
    Dim options As NameValueMap
    Set options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("SaveAssemSeperateFolder") = False
    options.Value("AssemDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportPoint") = True
    options.Value("ImportMeshes") = True
    options.Value("ImportAASP") = True
    options.Value("ImportAASPIndex") = 0
    options.Value("CreateSurfIndex") = 0
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
```

```
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = False
    options.Value("AdvanceHealing") = False
    options.Value("NoShowExpModelList") = True

    ' open exp file to get the model count
    ' prepare the 4th parameter for Open(), the final document

    Dim sourceObj As Object
    oTransAddIn.Open file, context, options, sourceObj

    Dim nModelCount As Integer
    nModelCount = options.Value("ExpModelCount")

    ' open specified model in exp file
    Dim i As Integer
    For i = 0 To nModelCount - 1
        options.Value("ExpModeIndex") = i
        oTransAddIn.Open file, context, options, sourceObj
    Next i
End Sub
```

Copy Code

```
    'Set CATIA V4 translator's CLSID and the file name (*.exp and *.dlv3).
    Dim strCLSID As String
    Dim strFileName As String
    strCLSID = "{C6ACD948-E1C5-4b5b-ADEE-3ED968F8CB1A}"
    strFileName = "C:\Hello.exp"

    Dim oAddIns As ApplicationAddIns
    oAddIns = ThisApplication.ApplicationAddIns

    ' find the translator and get CLSID and activate it
    Dim oTransAddIn As TranslatorAddIn
    oTransAddIn = oAddIns.ItemById(strCLSID)
    oTransAddIn.Activate

    ' get the transient object, take it as a factory to produce other objects
    Dim transientObj As TransientObjects
    transientObj = ThisApplication.TransientObjects

    ' prepare the 1st parameter for Open(), the file name
    Dim file As DataMedium
    file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' create an empty part
    Dim activeDoc As Document
    activeDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' prepare the 2nd parameter for Open()
    Dim context As TranslationContext
    context = transientObj.CreateTranslationContext
    context.Type = kFileBrowseIOMechanism
    ' this makes sure the imported stuff be inserted in the part doc created at the top
    context.OpenIntoExisting = activeDoc

    ' prepare the 3rd parameter for Open(), the options
    Dim options As NameValueMap
    options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("SaveAssemSeperateFolder") = False
    options.Value("AssemDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportPoint") = True
    options.Value("ImportMeshes") = True
    options.Value("ImportAASP") = True
    options.Value("ImportAASPIndex") = 0
    options.Value("CreateSurfIndex") = 0
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = False
    options.Value("AdvanceHealing") = False
    options.Value("NoShowExpModelList") = True

    ' open exp file to get the model count
    ' prepare the 4th parameter for Open(), the final document

    Dim sourceObj As Object
    oTransAddIn.Open(file, context, options, sourceObj)

    Dim nModelCount As Integer
    nModelCount = options.Value("ExpModelCount")

    ' open specified model in exp file
    Dim i As Integer
    For i = 0 To nModelCount - 1
        options.Value("ExpModeIndex") = i
        oTransAddIn.Open(file, context, options, sourceObj)
    Next i
```

Import DWG into sketch

Description

This sample demonstrates how to import DWG into sketch.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub ImportDWGIntoSketch()
    ' Get the DWG translator.
    Dim oDWGTranslator As TranslatorAddIn
    Set oDWGTranslator = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    Dim oDataMedium As DataMedium
    Set oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Specify a DWG file to import it into sketch
    oDataMedium.FileName = "C:\Temp\abc.dwg"

    Dim oTranslationContext As TranslationContext
    Set oTranslationContext = ThisApplication.TransientObjects.CreateTranslationContext
    oTranslationContext.Type = kFileBrowseIOMechanism

    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Get the sketch that the DWG will be imported into.
    Dim oSk As PlanarSketch
    Set oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    ' Open the sketch for edit.
    oSk.Edit

    ' Specify the sketch to import the DWG into.
    oTranslationContext.OpenIntoExisting = oSk

    Dim oOptions As NameValueMap
    Set oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Specify the layers to import
    oOptions.Add "SelectedLayers", "0"
    oOptions.Add "InvertLayersSelection", True

    ' Specify the units
    oOptions.Add "FileUnits", "Centimeters"

    ' Set to constraint the end points.
    oOptions.Add "ConstrainEndPoints", True

    ' Do the translation.
    Call oDWGTranslator.Open(oDataMedium, oTranslationContext, oOptions, oDoc)
End Sub
```

Copy Code

```
    ' Get the DWG translator.
    Dim oDWGTranslator As TranslatorAddIn
    oDWGTranslator = ThisApplication.ApplicationAddIns.ItemById("{C24E3AC2-122E-11D5-8E91-0010B541CD80}")

    Dim oDataMedium As DataMedium
    oDataMedium = ThisApplication.TransientObjects.CreateDataMedium

    ' Specify a DWG file to import it into sketch
    oDataMedium.FileName = "C:\Temp\abc.dwg"

    Dim oTranslationContext As TranslationContext
    oTranslationContext = ThisApplication.TransientObjects.CreateTranslationContext
    oTranslationContext.Type = kFileBrowseIOMechanism

    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Get the sketch that the DWG will be imported into.
    Dim oSk As PlanarSketch
    oSk = oDoc.ComponentDefinition.Sketches.Add(oDoc.ComponentDefinition.WorkPlanes(3))

    ' Open the sketch for edit.
    oSk.Edit

    ' Specify the sketch to import the DWG into.
    oTranslationContext.OpenIntoExisting = oSk

    Dim oOptions As NameValueMap
    oOptions = ThisApplication.TransientObjects.CreateNameValueMap

    ' Specify the layers to import
    oOptions.Add("SelectedLayers", "0")
    oOptions.Add("InvertLayersSelection", True)

    ' Specify the units
    oOptions.Add("FileUnits", "Centimeters")

    ' Set to constraint the end points.
    oOptions.Add("ConstrainEndPoints", True)

    ' Do the translation.
    Call oDWGTranslator.Open(oDataMedium, oTranslationContext, oOptions, oDoc)
```

Open an NX file suing the NX Translator Sample

Description

This sample demonstrates how open an NX file using the NX translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub ImportNXFile()
    ' Set NX translator's CLSID and the NX file name.
    Dim strCLSID As String
    Dim strFileName As String
    strCLSId = "{93D506C4-8355-4E28-9C4E-C2B5F1EDC6AE}"

    ' Please set the full file name here, such as "C:\Part1.prt"
    strFileName = "C:\Part1.prt "

    Dim oAddIns As ApplicationAddIns
    Set oAddIns = ThisApplication.ApplicationAddIns

    ' Find the NX translator, get the CLSID and activate it.
    Dim oTransAddIn As TranslatorAddIn
    Set oTransAddIn = oAddIns.ItemById(strCLSId)
    oTransAddIn.Activate

    ' Get the transient object and take it as a factory to produce other objects.
    Dim transientObj As TransientObjects
    Set transientObj = ThisApplication.TransientObjects

    ' Prepare the first parameter for Open(), the file name.
    Dim file As DataMedium
    Set file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    Set context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the third parameter for Open(), the options.
    Dim options As NameValueMap
    Set options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("SaveAssemSeperateFolder") = False
    options.Value("AssemDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportWorkPlane") = True
    options.Value("ImportWorkAxe") = True
    options.Value("ImportWorkPoint") = True
    options.Value("ImportPoint") = True
    options.Value("ImportAASP") = False
    options.Value("ImportAASPIndex") = 0
    options.Value("CreateSurfIndex") = 1
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = True
    options.Value("AdvanceHealing") = False

    options.Value("CHKSearchFolder") = True

    '100 is the search folder maximum that you can specify.
    'Assign separate search folder one by one.

    Dim searchFolder(100) As String
    searchFolder(0) = "C:\Folder1\"
    searchFolder(1) = "C:\Folder2\"
    options.Value("SearchFolder") = searchFolder

    'Prepare the fourth parameter for Open(), the final document
    Dim sourceObj As Object

    'Open the NX file.
    oTransAddIn.Open file, context, options, sourceObj
End Sub
```

Copy Code

```
    ' Set NX translator's CLSID and the NX file name.
    Dim strCLSID As String
    Dim strFileName As String
    strCLSId = "{93D506C4-8355-4E28-9C4E-C2B5F1EDC6AE}"

    ' Please set the full file name here, such as "C:\Part1.prt"
    strFileName = "C:\Part1.prt "

    Dim oAddIns As ApplicationAddIns
```

```
    oAddIns = ThisApplication.ApplicationAddIns

    ' Find the NX translator, get the CLSID and activate it.
    Dim oTransAddIn As TranslatorAddIn
    oTransAddIn = oAddIns.ItemById(strCLSId)
    oTransAddIn.Activate

    ' Get the transient object and take it as a factory to produce other objects.
    Dim transientObj As TransientObjects
    transientObj = ThisApplication.TransientObjects

    ' Prepare the first parameter for Open(), the file name.
    Dim file As DataMedium
    file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the third parameter for Open(), the options.
    Dim options As NameValueMap
    options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("SaveAssemSeperateFolder") = False
    options.Value("AssemDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportWorkPlane") = True
    options.Value("ImportWorkAxe") = True
    options.Value("ImportWorkPoint") = True
    options.Value("ImportPoint") = True
    options.Value("ImportAASP") = False
    options.Value("ImportAASPIndex") = 0
    options.Value("CreateSurfIndex") = 1
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = True
    options.Value("AdvanceHealing") = False

    options.Value("CHKSearchFolder") = True

    '100 is the search folder maximum that you can specify.
    'Assign separate search folder one by one.

    Dim searchFolder(100) As String
    searchFolder(0) = "C:\Folder1\"
    searchFolder(1) = "C:\Folder2\"
    options.Value("SearchFolder") = searchFolder

    'Prepare the fourth parameter for Open(), the final document
    Dim sourceObj As Object

    'Open the NX file.
    oTransAddIn.Open(file, context, options, sourceObj)
```

Import Revit data into Inventor

Description

The samples demonstrate how to import Revit data(.rvt) into Inventor part and assembly documents.

Code Samples

- VBA
- iLogic

The VBA samples demonstrate how to import Revit data(.rvt) into Inventor part and assembly documents. You should prepare a Revit data before running the samples.

Copy Code

```
Sub ImportRVTToAssembly()
    Dim oDoc As AssemblyDocument
    Set oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim sFile As String
    sFile = "C:\RevitFile.rvt"

    Dim oImportedRVTDef As ImportedRVTComponentDefinition
    Set oImportedRVTDef = oDoc.ComponentDefinition.ImportedComponents.CreateDefinition(sFile)
    oImportedRVTDef.Imported3DView = "{3D}"
    oImportedRVTDef.ImportedAssemblyOrganizationType = kImportedAsAssembly

    Dim oComp As ImportedRVTComponent
    Set oComp = oDoc.ComponentDefinition.ImportedComponents.Add(oImportedRVTDef)
End Sub

Sub ImportRVTToPart()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)
```

```
    Dim sFile As String
    sFile = "C:\RevitFile.rvt"

    Dim oImportedRVTDef As ImportedRVTComponentDefinition
    Set oImportedRVTDef = oDoc.ComponentDefinition.ReferenceComponents.ImportedComponents.CreateDefinition(sFile)

    oImportedRVTDef.Imported3DView = "{3D}"
    oImportedRVTDef.ImportedAssemblyOrganizationType = kImportedAsMultibodyPart

    Dim oComp As ImportedRVTComponent
    Set oComp = oDoc.ComponentDefinition.ReferenceComponents.ImportedComponents.Add(oImportedRVTDef)
End Sub
```

The VBA samples demonstrate how to import Revit data(.rvt) into Inventor part and assembly documents. You should prepare a Revit data before running the samples.

Copy Code

```
Sub Main
        ' Import to Assembly
        ImportRVTToAssembly()
        ' Import to Part
        ImportRVTToPart()

End Sub

Sub ImportRVTToAssembly()

    Dim oDoc As AssemblyDocument
    oDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim sFile As String
    sFile = "C:\RevitFile.rvt"

    Dim oImportedRVTDef As ImportedRVTComponentDefinition
    oImportedRVTDef = oDoc.ComponentDefinition.ImportedComponents.CreateDefinition(sFile)
    oImportedRVTDef.Imported3DView = "{3D}"
    oImportedRVTDef.ImportedAssemblyOrganizationType = kImportedAsAssembly

    Dim oComp As ImportedRVTComponent
    oComp = oDoc.ComponentDefinition.ImportedComponents.Add(oImportedRVTDef)
End Sub

Sub ImportRVTToPart()
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim sFile As String
    sFile = "C:\RevitFile.rvt"

    Dim oImportedRVTDef As ImportedRVTComponentDefinition
    oImportedRVTDef = oDoc.ComponentDefinition.ReferenceComponents.ImportedComponents.CreateDefinition(sFile)

    oImportedRVTDef.Imported3DView = "{3D}"
    oImportedRVTDef.ImportedAssemblyOrganizationType = kImportedAsMultibodyPart

    Dim oComp As ImportedRVTComponent
    oComp = oDoc.ComponentDefinition.ReferenceComponents.ImportedComponents.Add(oImportedRVTDef)
End Sub
```

Open Rhino Translator Sample

Description

This sample demonstrates how to opening a Rhino file using the Rhino translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub ImportRhino()
   ' Set Rhino translator's CLSID and the Rhino file name.
   Dim strCLSID As String
   Dim strFileName As String
   strCLSId = "{2CB23BF0-E2AC-4b32-B0A1-1CC292AF6623}"

   ' Please set the full file name here, such as "C:\ETTI.3dm"
   strFileName = "ETTI.3dm"

   Dim oAddIns As ApplicationAddIns
   Set oAddIns = ThisApplication.ApplicationAddIns

   ' Find the Rhino translator, get the CLSID and activate it.
   Dim oPeTransAddIn As TranslatorAddIn
   Set oPeTransAddIn = oAddIns.ItemById(strCLSId)
   oPeTransAddIn.Activate

   ' Get the transient object and take it as a factory to produce other objects
   Dim transientObj As TransientObjects
   Set transientObj = ThisApplication.TransientObjects

   ' Prepare the first parameter for Open(), the file name
   Dim file As DataMedium
   Set file = transientObj.CreateDataMedium
   file.FileName = strFileName
```

```
    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    Set context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the 3rd parameter for Open(), the options.
    Dim options As NameValueMap
    Set options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportPoint") = False
    options.Value("CreateSurfIndex") = 1
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
    options.Value("CEGroupLevel") = 0
    options.Value("CEPrefixCk") = False
    options.Value("CEPrefixString") = ""
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = True
    options.Value("AdvanceHealing") = False

    ' Prepare the fourth parameter for Open(), the final document.
    Dim sourceObj As Object

    ' Open the Rhino file.
    oPeTransAddIn.Open file, context, options, sourceObj
End Sub
```

Copy Code

```
     ' Set Rhino translator's CLSID and the Rhino file name.
    Dim strCLSID As String
    Dim strFileName As String
    strCLSId = "{2CB23BF0-E2AC-4b32-B0A1-1CC292AF6623}"

    ' Please set the full file name here, such as "C:\ETTI.3dm"
    strFileName = "ETTI.3dm"

    Dim oAddIns As ApplicationAddIns
    oAddIns = ThisApplication.ApplicationAddIns

    ' Find the Rhino translator, get the CLSID and activate it.
    Dim oPeTransAddIn As TranslatorAddIn
    oPeTransAddIn = oAddIns.ItemById(strCLSId)
    oPeTransAddIn.Activate

    ' Get the transient object and take it as a factory to produce other objects
    Dim transientObj As TransientObjects
    transientObj = ThisApplication.TransientObjects

    ' Prepare the first parameter for Open(), the file name
    Dim file As DataMedium
    file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the 3rd parameter for Open(), the options.
    Dim options As NameValueMap
    options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("ImportSolid") = True
    options.Value("ImportSurface") = True
    options.Value("ImportWire") = True
    options.Value("ImportPoint") = False
    options.Value("CreateSurfIndex") = 1
    options.Value("GroupNameIndex") = 0
    options.Value("GroupName") = ""
    options.Value("CEGroupLevel") = 0
    options.Value("CEPrefixCk") = False
    options.Value("CEPrefixString") = ""
    options.Value("ImportUnit") = 0
    options.Value("CheckDuringLoad") = False
    options.Value("AutoStitchAndPromote") = True
    options.Value("AdvanceHealing") = False

    ' Prepare the fourth parameter for Open(), the final document.
    Dim sourceObj As Object

    ' Open the Rhino file.
    oPeTransAddIn.Open(file, context, options, sourceObj)
```

Open an STL file using the STL Translator Sample

Description

This sample demonstrates how open an STL file using the STL translator add-in.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub ImportFunc()
    ' Set STL translator's CLSID and STL file name.
    Dim strCLSID As String
    Dim strFileName As String
    strCLSId = "{81CA7D27-2DBE-4058-8188-9136F85FC859}"
    strFileName = "C:\Part1.stl"

    Dim oAddIns As ApplicationAddIns
    Set oAddIns = ThisApplication.ApplicationAddIns

    ' Find the Rhino translator, get the CLSID and activate it.
    Dim oTransAddIn As TranslatorAddIn
    Set oTransAddIn = oAddIns.ItemById(strCLSId)
    oTransAddIn.Activate

    ' Get the transient object and take it as a factory to produce other objects
    Dim transientObj As TransientObjects
    Set transientObj = ThisApplication.TransientObjects

    ' Prepare the first parameter for Open(), the file name
    Dim file As DataMedium
    Set file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    Set context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the 3rd parameter for Open(), the options.
    Dim options As NameValueMap
    Set options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("ImportUnit") = 1
    options.Value("ImportColor") = True
    options.Value("ImportColorIndex") = 0

    ' Prepare the fourth parameter for Open(), the final document.
    Dim sourceObj As Object

    ' Open the STL file.
    oTransAddIn.Open file, context, options, sourceObj
End Sub
```

Copy Code

```
    ' Set STL translator's CLSID and STL file name.
    Dim strCLSID As String
    Dim strFileName As String
    strCLSId = "{81CA7D27-2DBE-4058-8188-9136F85FC859}"
    strFileName = "C:\Part1.stl"

    Dim oAddIns As ApplicationAddIns
    oAddIns = ThisApplication.ApplicationAddIns

    ' Find the Rhino translator, get the CLSID and activate it.
    Dim oTransAddIn As TranslatorAddIn
    oTransAddIn = oAddIns.ItemById(strCLSId)
    oTransAddIn.Activate

    ' Get the transient object and take it as a factory to produce other objects
    Dim transientObj As TransientObjects
    transientObj = ThisApplication.TransientObjects

    ' Prepare the first parameter for Open(), the file name
    Dim file As DataMedium
    file = transientObj.CreateDataMedium
    file.FileName = strFileName

    ' Prepare the second parameter for Open(), the open type.
    Dim context As TranslationContext
    context = transientObj.CreateTranslationContext
    context.Type = kDataDropIOMechanism

    ' Prepare the 3rd parameter for Open(), the options.
    Dim options As NameValueMap
    options = transientObj.CreateNameValueMap
    options.Value("SaveComponentDuringLoad") = False
    options.Value("SaveLocationIndex") = 0
    options.Value("ComponentDestFolder") = ""
    options.Value("ImportUnit") = 1
    options.Value("ImportColor") = True
    options.Value("ImportColorIndex") = 0

    ' Prepare the fourth parameter for Open(), the final document.
    Dim sourceObj As Object

    ' Open the STL file.
    oTransAddIn.Open(file, context, options, sourceObj)
```

InteractionGraphics

Description

The sample creates overlay graphics.

Code Samples

- VBA
- iLogic

Follow these steps: 1. Have a drawing document open. 2. Cut and paste in a VBA module. 3. Run the sample. Overlay graphics is created. 4. Hit the escape key to end interaction events and bring down the graphics.

Copy Code

```
Dim oIE As InteractionEvents


Public Sub DrawOverlayGraphics()
    Dim oDoc As Document
    Set oDoc = ThisApplication.ActiveDocument

    Set oIE = ThisApplication.CommandManager.CreateInteractionEvents
    oIE.Start

    Dim oIG As InteractionGraphics
    Set oIG = oIE.InteractionGraphics

    On Error Resume Next
    Dim oDataSets As GraphicsDataSets
    Set oDataSets = oIG.GraphicsDataSets

    ' Set a reference to the transient geometry object for use later.
    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Create a coordinate set.
    Dim oCoordSet As GraphicsCoordinateSet
    Set oCoordSet = oDataSets.CreateCoordinateSet(1)

    ' Create an array that contains coordinates that define a set
    ' of outwardly spiraling points.
    Dim oPointCoords(1 To 90) As Double
    Dim i As Long
    Dim dRadius As Double
    dRadius = 1
    Dim dAngle As Double
    For i = 0 To 29
        ' Define the X, Y, and Z components of the point.
        oPointCoords(i * 3 + 1) = dRadius * Cos(dAngle)
        oPointCoords(i * 3 + 2) = dRadius * Sin(dAngle)
        oPointCoords(i * 3 + 3) = i / 2

        ' Increment the angle and radius to create the spiral.
        dRadius = dRadius + 0.25
        dAngle = dAngle + (3.14159265358979 / 6)
    Next

    ' Assign the points into the coordinate set.
    Call oCoordSet.PutCoordinates(oPointCoords)

    ' Create the ClientGraphics object.
    Dim oClientGraphics As ClientGraphics
    Set oClientGraphics = oIG.OverlayClientGraphics

    ' Create a new graphics node within the client graphics objects.
    Dim oLineNode As GraphicsNode
    Set oLineNode = oClientGraphics.AddNode(1)

    ' Create a LineGraphics object within the node.
    Dim oLineSet As LineGraphics
    Set oLineSet = oLineNode.AddLineGraphics

    ' Assign the coordinate set to the line graphics.
    oLineSet.CoordinateSet = oCoordSet

    ' Update the view to see the resulting spiral.
    oIG.UpdateOverlayGraphics ThisApplication.ActiveView

    ' Create another graphics node for a line strip.
    Dim oLineStripNode As GraphicsNode
    Set oLineStripNode = oClientGraphics.AddNode(2)

    ' Create a LineStripGraphics object within the new node.
    Dim oLineStrip As LineStripGraphics
    Set oLineStrip = oLineStripNode.AddLineStripGraphics

    ' Assign the same coordinate set to the line strip.
    oLineStrip.CoordinateSet = oCoordSet

    ' Create a color set to use in defining a explicit color to the line strip.
    Dim oColorSet As GraphicsColorSet
    Set oColorSet = oDataSets.CreateColorSet(1)

    ' Add a single color to the set that is red.
    Call oColorSet.Add(1, 255, 0, 0)

    ' Assign the color set to the line strip.
    oLineStrip.ColorSet = oColorSet

    ' The two spirals are currently on top of each other so translate the
```

```
    ' new one in the x direction so they're side by side.
    Dim oMatrix As Matrix
    Set oMatrix = oLineStripNode.Transformation
    Call oMatrix.SetTranslation(oTransGeom.CreateVector(15, 0, 0))
    oLineStripNode.Transformation = oMatrix

    ' Update the view to see the resulting spiral.
    oIG.UpdateOverlayGraphics ThisApplication.ActiveView

    'oIE.Stop
End Sub
```

Follow these steps: 1. Have a drawing document open. 2. Copy the code to iLogic rule. 3. Run the sample. Overlay graphics is created. 4. Hit the escape key to end interaction events and bring down the graphics.

Copy Code

```
Class Test
        Dim oIE As InteractionEvents

        Sub Main
            Dim oDoc As Document
            oDoc = ThisApplication.ActiveDocument

            oIE = ThisApplication.CommandManager.CreateInteractionEvents
            oIE.Start

            Dim oIG As InteractionGraphics
            oIG = oIE.InteractionGraphics

            On Error Resume Next
            Dim oDataSets As GraphicsDataSets
            oDataSets = oIG.GraphicsDataSets

            ' Set a reference to the transient geometry object for use later.
            Dim oTransGeom As TransientGeometry
            oTransGeom = ThisApplication.TransientGeometry

            ' Create a coordinate set.
            Dim oCoordSet As GraphicsCoordinateSet
            oCoordSet = oDataSets.CreateCoordinateSet(1)

            ' Create an array that contains coordinates that define a set
            ' of outwardly spiraling points.
            Dim oPointCoords(0 To 89) As Double
            Dim i As Long
            Dim dRadius As Double
            dRadius = 1
            Dim dAngle As Double
            For i = 0 To 29
                ' Define the X, Y, and Z components of the point.
                oPointCoords(i * 3) = dRadius * Cos(dAngle)
                oPointCoords(i * 3 + 1) = dRadius * Sin(dAngle)
                oPointCoords(i * 3 + 2) = i / 2

                ' Increment the angle and radius to create the spiral.
                dRadius = dRadius + 0.25
                dAngle = dAngle + (3.14159265358979 / 6)
            Next

            ' Assign the points into the coordinate set.
            Call oCoordSet.PutCoordinates(oPointCoords)

            ' Create the ClientGraphics object.
            Dim oClientGraphics As ClientGraphics
            oClientGraphics = oIG.OverlayClientGraphics

            ' Create a new graphics node within the client graphics objects.
            Dim oLineNode As GraphicsNode
            oLineNode = oClientGraphics.AddNode(1)

            ' Create a LineGraphics object within the node.
            Dim oLineSet As LineGraphics
            oLineSet = oLineNode.AddLineGraphics

            ' Assign the coordinate set to the line graphics.
            oLineSet.CoordinateSet = oCoordSet

            ' Update the view to see the resulting spiral.
            oIG.UpdateOverlayGraphics(ThisApplication.ActiveView)

            ' Create another graphics node for a line strip.
            Dim oLineStripNode As GraphicsNode
            oLineStripNode = oClientGraphics.AddNode(2)

            ' Create a LineStripGraphics object within the new node.
            Dim oLineStrip As LineStripGraphics
            oLineStrip = oLineStripNode.AddLineStripGraphics

            ' Assign the same coordinate set to the line strip.
            oLineStrip.CoordinateSet = oCoordSet

            ' Create a color set to use in defining a explicit color to the line strip.
            Dim oColorSet As GraphicsColorSet
            oColorSet = oDataSets.CreateColorSet(1)

            ' Add a single color to the set that is red.
            Call oColorSet.Add(1, 255, 0, 0)

            ' Assign the color set to the line strip.
            oLineStrip.ColorSet = oColorSet

            ' The two spirals are currently on top of each other so translate the
            ' new one in the x direction so they're side by side.
            Dim oMatrix As Matrix
            oMatrix = oLineStripNode.Transformation
            Call oMatrix.SetTranslation(oTransGeom.CreateVector(15, 0, 0))
```

```
            oLineStripNode.Transformation = oMatrix

            ' Update the view to see the resulting spiral.
            oIG.UpdateOverlayGraphics(ThisApplication.ActiveView)

            'oIE.Stop
        End Sub
End Class
```

Creating a HighlightSet

Description

Demonstrates creating a highlight set.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub Main()
  ' Check to make sure the active document is a part.
  If ThisApplication.ActiveDocumentType <> kPartDocumentObject Then
     MsgBox "A part document must be open."
     End
  End If

  ' Set a reference to the active part document.
  Dim oDoc As PartDocument
  Set oDoc = ThisApplication.ActiveDocument

  ' Arbitrarily get two faces.  (Assumes a model exists that has
  ' at least 3 faces.
  Dim oFace1 As Face
  Dim oFace2 As Face
  Set oFace1 = oDoc.ComponentDefinitions(1).SurfaceBodies(1).Faces(1)
  Set oFace2 = oDoc.ComponentDefinitions(1).SurfaceBodies(1).Faces(3)

  ' Create a highlight set.
  Dim oSet1 As HighlightSet
  Set oSet1 = oDoc.CreateHighlightSet

  ' Add the first face to the highlight set.
   Call oSet1.AddItem(oFace1)

  ' Create another highlight set.
  Dim oSet2 As HighlightSet
  Set oSet2 = oDoc.CreateHighlightSet

  ' Change the color of the highlight set to green.
  oSet2.Color = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

  ' Add the second face to the highlight set.
  Call oSet2.AddItem(oFace2)
End Sub
```

Copy Code

```
    ' Check to make sure the active document is a part.
  If ThisApplication.ActiveDocumentType <> kPartDocumentObject Then
     MsgBox("A part document must be open.")
     Exit Sub
  End If

  ' Set a reference to the active part document.
  Dim oDoc As PartDocument
  oDoc = ThisApplication.ActiveDocument

  ' Arbitrarily get two faces.  (Assumes a model exists that has
  ' at least 3 faces.
  Dim oFace1 As Face
  Dim oFace2 As Face
  oFace1 = oDoc.ComponentDefinitions(1).SurfaceBodies(1).Faces(1)
  oFace2 = oDoc.ComponentDefinitions(1).SurfaceBodies(1).Faces(3)

  ' Create a highlight set.
  Dim oSet1 As HighlightSet
  oSet1 = oDoc.CreateHighlightSet

  ' Add the first face to the highlight set.
   Call oSet1.AddItem(oFace1)

  ' Create another highlight set.
  Dim oSet2 As HighlightSet
  oSet2 = oDoc.CreateHighlightSet

  ' Change the color of the highlight set to green.
  oSet2.Color = ThisApplication.TransientObjects.CreateColor(0, 255, 0)

  ' Add the second face to the highlight set.
  Call oSet2.AddItem(oFace2)
```

Using measure events

Description

This sample demonstrates using the measure events to measure distance and angle. Interactive measure is dependent on events and VB only supports events within a class module. To use the sample copy the InteractiveMeasureDistance and InteractiveMeasureAngle subs into a code module. Create a new class module called clsMeasure and copy all of the rest of the code into it.

Code Samples

- VBA
- iLogic

To run the sample, have a document open that contains some geometry and run the InteractiveMeasureDistance and InteractiveMeasureAngle subs.

Copy Code

```
Public Sub InteractiveMeasureDistance()
    ' Create a new clsMeasure object.
    Dim oMeasure As New clsMeasure

    ' Call the Measure method of the clsMeasure object
    Call oMeasure.Measure(kDistanceMeasure)
End Sub


Public Sub InteractiveMeasureAngle()
    ' Create a new clsMeasure object.
    Dim oMeasure As New clsMeasure

    ' Call the Measure method of the clsMeasure object
    Call oMeasure.Measure(kAngleMeasure)
End Sub


'**********************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsMeasure". The name can be
' changed but you'll need to change the declaration in the
' calling function "InteractiveMeasureDistance" and
' "InteractiveMeasureAngle" to use the new name.

' Declare the event objects
Private WithEvents oInteractEvents As InteractionEvents
Private WithEvents oMeasureEvents As MeasureEvents

' Declare a flag that's used to determine when measuring stops.
Private bStillMeasuring As Boolean
Private eMeasureType As MeasureTypeEnum

Public Sub Measure(MeasureType As MeasureTypeEnum)

    eMeasureType = MeasureType

    ' Initialize flag.
    bStillMeasuring = True

    ' Create an InteractionEvents object.
    Set oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

    ' Set a reference to the measure events.
    Set oMeasureEvents = oInteractEvents.MeasureEvents
    oMeasureEvents.Enabled = True

    ' Start the InteractionEvents object.
    oInteractEvents.Start

    ' Start measure tool
    If eMeasureType = kDistanceMeasure Then
        oMeasureEvents.Measure kDistanceMeasure
    Else
        oMeasureEvents.Measure kAngleMeasure
    End If

    ' Loop until a selection is made.
    Do While bStillMeasuring
        DoEvents
    Loop

    ' Stop the InteractionEvents object.
    oInteractEvents.Stop

    ' Clean up.
    Set oMeasureEvents = Nothing
    Set oInteractEvents = Nothing
End Sub

Private Sub oInteractEvents_OnTerminate()
    ' Set the flag to indicate we're done.
    bStillMeasuring = False
End Sub

Private Sub oMeasureEvents_OnMeasure(ByVal MeasureType As MeasureTypeEnum, ByVal MeasuredValue As Double, ByVal Context As NameValueMa

    Dim strMeasuredValue As String

    If eMeasureType = kDistanceMeasure Then
        strMeasureValue = ThisApplication.ActiveDocument.UnitsOfMeasure.GetStringFromValue(MeasuredValue, kDefaultDisplayLengthUnits)
        MsgBox "Distance = " & strMeasureValue, vbOKOnly, "Measure Distance"
    Else
        strMeasureValue = ThisApplication.ActiveDocument.UnitsOfMeasure.GetStringFromValue(MeasuredValue, kDefaultDisplayAngleUnits)
        MsgBox "Angle = " & strMeasureValue, vbOKOnly, "Measure Angle"
```

```
    End If

    ' Set the flag to indicate we're done.
    bStillMeasuring = False

End Sub
```

To run the sample, have a document open that contains some geometry and run the InteractiveMeasureDistance and InteractiveMeasureAngle subs.

Copy Code

```
Sub Main
        ' Measure distance
        InteractiveMeasureDistance()

        ' Measure angle
        InteractiveMeasureAngle
End Sub

Public Sub InteractiveMeasureDistance()
        ' Create a new clsMeasure object.
    Dim oMeasure As New clsMeasure
        oMeasure.Init(ThisApplication)

    ' Call the Measure method of the clsMeasure object
    Call oMeasure.Measure(kDistanceMeasure)
End Sub


Public Sub InteractiveMeasureAngle()
    ' Create a new clsMeasure object.
    Dim oMeasure As New clsMeasure
        oMeasure.Init(ThisApplication)

    ' Call the Measure method of the clsMeasure object
    Call oMeasure.Measure(kAngleMeasure)
End Sub


Class clsMeasure
        ' Declare the event objects
        Private WithEvents oInteractEvents As InteractionEvents
        Private WithEvents oMeasureEvents As MeasureEvents
        Private ThisApplication As Inventor.Application

        ' Declare a flag that's used to determine when measuring stops.
        Private bStillMeasuring As Boolean
        Private eMeasureType As MeasureTypeEnum

        Public Sub Init(oApp As Inventor.Application)
                ThisApplication = oApp
        End Sub

        Public Sub Measure(MeasureType As MeasureTypeEnum)

            eMeasureType = MeasureType

            ' Initialize flag.
            bStillMeasuring = True

            ' Create an InteractionEvents object.
            oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

            ' Set a reference to the measure events.
            oMeasureEvents = oInteractEvents.MeasureEvents
            oMeasureEvents.Enabled = True

            ' Start the InteractionEvents object.
            oInteractEvents.Start

            ' Start measure tool
            If eMeasureType = kDistanceMeasure Then
                oMeasureEvents.Measure(kDistanceMeasure)
            Else
                oMeasureEvents.Measure(kAngleMeasure)
            End If

            ' Loop until a selection is made.
            Do While bStillMeasuring
                ThisApplication.UserInterfaceManager.DoEvents
            Loop

            ' Stop the InteractionEvents object.
            oInteractEvents.Stop

            ' Clean up.
            oMeasureEvents = Nothing
            oInteractEvents = Nothing
        End Sub

        Private Sub oInteractEvents_OnTerminate()
            ' Set the flag to indicate we're done.
            bStillMeasuring = False
        End Sub

        Private Sub oMeasureEvents_OnMeasure(ByVal MeasureType As MeasureTypeEnum, ByVal MeasuredValue As Double, ByVal Context As Nam

            Dim strMeasuredValue As String

            If eMeasureType = kDistanceMeasure Then
                strMeasureValue = ThisApplication.ActiveDocument.UnitsOfMeasure.GetStringFromValue(MeasuredValue, kDefaultDisplayLengt
                MsgBox("Distance = " & strMeasureValue, vbOKOnly, "Measure Distance")
            Else
                strMeasureValue = ThisApplication.ActiveDocument.UnitsOfMeasure.GetStringFromValue(MeasuredValue, kDefaultDisplayAngle
                MsgBox("Angle = " & strMeasureValue, vbOKOnly, "Measure Angle")
            End If

            ' Set the flag to indicate we're done.
```

```
            bStillMeasuring = False

        End Sub
End Class
```

Post Private Event Sample

Description

This sample demonstrates how to use the PostPrivateEvent to configure the options for placing a part component.

Code Samples

- VBA
- iLogic

This sample demonstrates how to use the PostPrivateEvent to configure the options for placing a part component.

Copy Code

```
Sub AssemblyPlaceComponentCmdWithPPE()
    ' In UI create a part and create an iMate definition to its face, save it and set its full filename below.

    Dim strFullFileName As String
    strFullFileName = "C:\Temp\iMate.ipt"

    Dim oAssyDoc As AssemblyDocument
    Set oAssyDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)


    Dim oOccu As ComponentOccurrence
    Set oOccu = oAssyDoc.ComponentDefinition.Occurrences.Add(strFullFileName, ThisApplication.TransientGeometry.CreateMatrix)

    Dim oCM As CommandManager
    Set oCM = ThisApplication.CommandManager

    ' clear existing clipboard
    oCM.ClearPrivateEvents

    Call oCM.PostPrivateEvent(kFileNameEvent, strFullFileName)

    Dim oNV As NameValueMap
    Set oNV = ThisApplication.TransientObjects.CreateNameValueMap

    Call oNV.Add("Use iMate", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)

    oNV.Clear
    Call oNV.Add("Generate iMate Results", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)

    Call oCM.PostPrivateEvent(kStringEvent, "Design View Representation|Front")

    oNV.Clear
    Call oNV.Add("Associative Design View", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)


    Call oCM.PostPrivateEvent(kStringEvent, "Model State|[Primary]")

    ' Call below command will cause to placing the component using above settings.
    Dim oCat As ControlDefinition
    Set oCat = oCM.ControlDefinitions.Item("AssemblyPlaceComponentCmd")
    Call oCat.Execute2(True)
End Sub
```

This sample demonstrates how to use the PostPrivateEvent to configure the options for placing a part component.

Copy Code

```
    ' In UI create a part and create an iMate definition to its face, save it and set its full filename below.

    Dim strFullFileName As String
    strFullFileName = "C:\Temp\iMate.ipt"

    Dim oAssyDoc As AssemblyDocument
    oAssyDoc = ThisApplication.Documents.Add(kAssemblyDocumentObject)

    Dim oOccu As ComponentOccurrence
    oOccu = oAssyDoc.ComponentDefinition.Occurrences.Add(strFullFileName, ThisApplication.TransientGeometry.CreateMatrix)

    Dim oCM As CommandManager
    oCM = ThisApplication.CommandManager

    ' clear existing clipboard
    oCM.ClearPrivateEvents

    Call oCM.PostPrivateEvent(kFileNameEvent, strFullFileName)

    Dim oNV As NameValueMap
    oNV = ThisApplication.TransientObjects.CreateNameValueMap

    Call oNV.Add("Use iMate", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)

    oNV.Clear
    Call oNV.Add("Generate iMate Results", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)
```

```
    Call oCM.PostPrivateEvent(kStringEvent, "Design View Representation|Front")

    oNV.Clear
    Call oNV.Add("Associative Design View", True)
    Call oCM.PostPrivateEvent(kBooleanEvent, oNV)


    Call oCM.PostPrivateEvent(kStringEvent, "Model State|[Primary]")

    ' Call below command will cause to placing the component using above settings.
    Dim oCat As ControlDefinition
    oCat = oCM.ControlDefinitions.Item("AssemblyPlaceComponentCmd")
    Call oCat.Execute2(True)
```

Prompt message creation sample

Description

This sample demonstrates how to creat a custom prompt message.

Code Samples

- VBA
- iLogic

This sample demonstrates how to creat a custom prompt message.

Copy Code

```
Sub PromptMessageSample()
    Dim oPO As PromptsOptions
    Set oPO = ThisApplication.PromptsOptions

    Dim oPMs As PromptMessages
    Set oPMs = oPO.PromptMessages

    Dim oPM As PromptMessage

    On Error Resume Next
    Set oPM = oPMs.Item("User_defined_prompt")

    If Err Then
        Set oPM = oPMs.Add("User_defined_prompt", "This is not allowed", vbOKCancel, vbCritical, "Inventor", vbDefaultButton1, kNoRest
    End If

    On Error GoTo 0

    oPM.Display
End Sub
```

This sample demonstrates how to creat a custom prompt message.

Copy Code

```
    Dim oPO As PromptsOptions
    oPO = ThisApplication.PromptsOptions

    Dim oPMs As PromptMessages
    oPMs = oPO.PromptMessages

    Dim oPM As PromptMessage

    On Error Resume Next
    oPM = oPMs.Item("User_defined_prompt")

    If Err.Number > 0 Then
        oPM = oPMs.Add("User_defined_prompt", "This is not allowed", vbOKCancel, vbCritical, "Inventor", vbDefaultButton1, kNoRestrict
    End If

    On Error GoTo 0
    ThisApplication.SilentOperation = False
    oPM.Display()
```

WebBrowserDialog creation

Description

This sample demonstrates how to create a web-based browser dialog, you can use this dialog to show a html format file or navigate to a website etc..

Code Samples

- VBA
- iLogic

Copy Code

```
Sub WebBrowserDialogSample()
    ' Create a WebBrowserDialog
    Dim oWebBrowserDialog As WebBrowserDialog
```

```
    Set oWebBrowserDialog = ThisApplication.WebBrowserDialogs.Add("MyBrowser")
    oWebBrowserDialog.WindowState = kNormalWindow

    ' Nagigate to a web site
    Call oWebBrowserDialog.Navigate("http://www.autodesk.com")

    ' Play a tutorial video if you have the Interactive Tutorial installed
    ' Call oWebBrowserDialog.Navigate("C:\Users\Public\Documents\Autodesk\Inventor 2017\Interactive Tutorial\en-US\Fundamentals\Video\I

    ' Delete it
    oWebBrowserDialog.Delete
End Sub
```

Copy Code

```
    ' Create a WebBrowserDialog
    Dim oWebBrowserDialog As WebBrowserDialog
    oWebBrowserDialog = ThisApplication.WebBrowserDialogs.Add("MyBrowser")
    oWebBrowserDialog.WindowState = kNormalWindow

    ' Nagigate to a web site
    Call oWebBrowserDialog.Navigate("http://www.autodesk.com")

    ' Play a tutorial video if you have the Interactive Tutorial installed
    ' Call oWebBrowserDialog.Navigate("C:\Users\Public\Documents\Autodesk\Inventor 2017\Interactive Tutorial\en-US\Fundamentals\Video\I

    ' Delete it
    oWebBrowserDialog.Delete
```

Add commands to the application menu

Description

Demonstrates adding command to the application menu.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub AddCommandsToFileBrowser()
    ' Get the application menu controls collection
    Dim oFileBrowserControls As CommandControls
    Set oFileBrowserControls = ThisApplication.UserInterfaceManager.FileBrowserControls

    ' Get the "Zoom All" and "Home View" commands
    Dim oDef1 As ButtonDefinition
    Set oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("AppZoomAllCmd")

    Dim oDef2 As ButtonDefinition
    Set oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AppIsometricViewCmd")

    ' Create button controls, positioned before the "Manage" control
    Call oFileBrowserControls.AddButton(oDef1, True, True, "Manage", True)
    Call oFileBrowserControls.AddButton(oDef2, True, True, "Manage", True)
    Call oFileBrowserControls.AddSeparator("Manage", True)
End Sub
```

Copy Code

```
    ' Get the application menu controls collection
    Dim oFileBrowserControls As CommandControls
    oFileBrowserControls = ThisApplication.UserInterfaceManager.FileBrowserControls

    ' Get the "Zoom All" and "Home View" commands
    Dim oDef1 As ButtonDefinition
    oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("AppZoomAllCmd")

    Dim oDef2 As ButtonDefinition
    oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AppIsometricViewCmd")

    ' Create button controls, positioned before the "Manage" control
    Call oFileBrowserControls.AddButton(oDef1, True, True, "Manage", True)
    Call oFileBrowserControls.AddButton(oDef2, True, True, "Manage", True)
    Call oFileBrowserControls.AddSeparator("Manage", True)
```

Adjust the brightness of lighting

Description

This sample demonstrates how to adjust lighting brightness with mini-toolbar slider.

Code Samples

- VBA

- iLogic

Copy Code

```
Sub BrightnessAdjustSample()
    ' Open a part/assembly document before running below code
    If Not (ThisApplication.ActiveDocument Is Nothing) Then
        If ThisApplication.ActiveDocument.DocumentType = kPartDocumentObject Or ThisApplication.ActiveDocument.DocumentType = kAssembl
            Dim ocls As New clsBrightnessAdjust
            ocls.Init
        Else
            MsgBox "Please activate/open a part or assembly for the demo!"
        End If
    Else
        MsgBox "Please open a part or assembly document first for the demo!"
    End If
End Sub

'**********************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsBrightnessAdjust".  The name
' can be changed but you'll need to change the declaration in
' the calling function "BrightnessAdjustSample" to use the new name.
Private WithEvents m_MiniToolbar As MiniToolbar
Private WithEvents m_Brightness As MiniToolbarSlider
Private WithEvents m_AppEvents As ApplicationEvents

Private m_BrightAdjustTransaction As Transaction
Private m_LastBrightnessVal As Double

Private m_Doc As Document
Private m_ActiveLightingStyle As LightingStyle
Private bStop As Boolean
Private bInitSliderVal As Boolean

Public Sub Init()
    Set m_AppEvents = ThisApplication.ApplicationEvents

    Set m_Doc = ThisApplication.ActiveDocument
    Set m_ActiveLightingStyle = m_Doc.ActiveLightingStyle

    Set m_MiniToolbar = ThisApplication.CommandManager.CreateMiniToolbar
    Set m_Brightness = m_MiniToolbar.Controls.AddSlider("Light_Brightness", "Brightness", "Adjust light brightness", kDoubleType, 1, 0
    m_Brightness.AutoHide = True

    ' Hide the Apply button.
    m_MiniToolbar.ShowApply = False

    ' Display the minitoobar and place it to the top-left of the view.
    m_MiniToolbar.Visible = True
    m_MiniToolbar.Position = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)

    bInitSliderVal = True

    ' Remember the last brightness value
    m_LastBrightnessVal = m_ActiveLightingStyle.Brightness

    bStop = False

    Set m_BrightAdjustTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Adjust brightness")

    Do
        If bInitSliderVal Then
            m_Brightness.Value = m_ActiveLightingStyle.Brightness
        End If
        ThisApplication.UserInterfaceManager.DoEvents
    Loop Until bStop

End Sub

Private Sub m_Brightness_OnValueChange()
    ' Ignore this event when the MiniToolbar value change is initialized by switching active LightingStyle
    If Not bInitSliderVal Then
        'bChangedBySlider = True
        m_ActiveLightingStyle.Brightness = m_Brightness.Value
    End If
    bInitSliderVal = False
    m_Doc.Views(1).Update

End Sub

' Cancel the brightness change
Private Sub m_MiniToolbar_OnCancel()
    bStop = True
    m_BrightAdjustTransaction.Abort
    m_Doc.Views(1).Update
End Sub

' If the brightness is changed, update the LightingStyle for this value.
Private Sub m_MiniToolbar_OnOK()
    bStop = True

    If m_LastBrightnessVal = m_ActiveLightingStyle.Brightness Then
        m_BrightAdjustTransaction.Abort
        m_Doc.Views(1).Update
    Else
        m_BrightAdjustTransaction.End
        m_Doc.Views(1).Update
    End If
End Sub
```

Copy Code

```
Sub Main
    ' Open a part/assembly document before running below code
    If Not (ThisApplication.ActiveDocument Is Nothing) Then
```

```
        If ThisApplication.ActiveDocument.DocumentType = kPartDocumentObject Or ThisApplication.ActiveDocument.DocumentType = kAssembl
            Dim ocls As New clsBrightnessAdjust

            ocls.Init(ThisApplication)
        Else
            MsgBox("Please activate/open a part or assembly for the demo!")
        End If
    Else
        MsgBox("Please open a part or assembly document first for the demo!")
    End If
End Sub

Class clsBrightnessAdjust
        Private WithEvents m_MiniToolbar As MiniToolbar
        Private WithEvents m_Brightness As MiniToolbarSlider
        Private WithEvents m_AppEvents As ApplicationEvents

        Private m_BrightAdjustTransaction As Transaction
        Private m_LastBrightnessVal As Double

        Private m_Doc As Document
        Private m_ActiveLightingStyle As LightingStyle
        Private bStop As Boolean
        Private bInitSliderVal As Boolean
        Private ThisApplication As Inventor.Application

        Public Sub Init(oApp As Inventor.Application)
                ThisApplication = oApp
            m_AppEvents = ThisApplication.ApplicationEvents

            m_Doc = ThisApplication.ActiveDocument
            m_ActiveLightingStyle = m_Doc.ActiveLightingStyle

            m_MiniToolbar = ThisApplication.CommandManager.CreateMiniToolbar
            m_Brightness = m_MiniToolbar.Controls.AddSlider("Light_Brightness", "Brightness", "Adjust light brightness", kDoubleType,
            m_Brightness.AutoHide = True

            ' Hide the Apply button.
            m_MiniToolbar.ShowApply = False

            ' Display the minitoobar and place it to the top-left of the view.
            m_MiniToolbar.Visible = True
            m_MiniToolbar.Position = ThisApplication.TransientGeometry.CreatePoint2d(0, 0)

            bInitSliderVal = True

            ' Remember the last brightness value
            m_LastBrightnessVal = m_ActiveLightingStyle.Brightness

            bStop = False

            m_BrightAdjustTransaction = ThisApplication.TransactionManager.StartTransaction(m_Doc, "Adjust brightness")

            Do
                If bInitSliderVal Then
                    m_Brightness.Value = m_ActiveLightingStyle.Brightness
                End If
                ThisApplication.UserInterfaceManager.DoEvents
            Loop Until bStop

        End Sub

        Private Sub m_Brightness_OnValueChange() Handles m_Brightness.OnValueChange
            ' Ignore this event when the MiniToolbar value change is initialized by switching active LightingStyle
            If Not bInitSliderVal Then
                'bChangedBySlider = True
                m_ActiveLightingStyle.Brightness = m_Brightness.Value
            End If
            bInitSliderVal = False
            m_Doc.Views(1).Update

        End Sub

        ' Cancel the brightness change
        Private Sub m_MiniToolbar_OnCancel() Handles m_MiniToolbar.OnCancel
            bStop = True
            m_BrightAdjustTransaction.Abort
            m_Doc.Views(1).Update
        End Sub

        ' If the brightness is changed, update the LightingStyle for this value.
        Private Sub m_MiniToolbar_OnOK() Handles m_MiniToolbar.OnOK
            bStop = True

            If m_LastBrightnessVal = m_ActiveLightingStyle.Brightness Then
                m_BrightAdjustTransaction.Abort
                m_Doc.Views(1).Update
            Else
                m_BrightAdjustTransaction.End
                m_Doc.Views(1).Update
            End If
        End Sub
End Class
```

Navigation between browser and data

Description

This sample demonstrates the navigation between a browser node and it's corresponding data model object and vice versa. This sample creates a work plane, finds its browser node and gets the work plane object back from the browser node.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub DataModelToBrowser()
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    ' Create a new workplane parallel to the XY plane.
    Dim oWorkPlane As WorkPlane
    Set oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes.Item(3), 1)

    ' Get the browser node definition associated with the work plane.
    Dim oNativeBrowserNodeDef As NativeBrowserNodeDefinition
    Set oNativeBrowserNodeDef = oPartDoc.BrowserPanes.GetNativeBrowserNodeDefinition(oWorkPlane)

    ' Get the top browser node of the model pane.
    Dim oTopBrowserNode As BrowserNode
    Set oTopBrowserNode = oPartDoc.BrowserPanes.ActivePane.TopNode

    ' Get the work plane browser node.
    ' This assumes that only one node references the browser node definition.
    ' An example of multiple nodes referencing a single definition is a shared
    ' sketch. The browser may have multiple nodes that represent the same shared
    ' sketch, but all of them reference the same definition.
    Dim oWorkPlaneNode As BrowserNode
    Set oWorkPlaneNode = oTopBrowserNode.AllReferencedNodes(oNativeBrowserNodeDef).Item(1)

    ' Get the browser node definition from the browser node.
    Set oNativeBrowserNodeDef = Nothing
    Set oNativeBrowserNodeDef = oWorkPlaneNode.BrowserNodeDefinition

    ' Get the work plane from the browser node definition.
    Set oWorkPlane = Nothing
    Set oWorkPlane = oNativeBrowserNodeDef.NativeObject

    ' Select the work plane to make sure we have the right object.
    oPartDoc.SelectSet.Select oWorkPlane
End Sub
```

Copy Code

```
    ' Create a new part document, using the default part template.
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' Set a reference to the component definition.
    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    ' Create a new workplane parallel to the XY plane.
    Dim oWorkPlane As WorkPlane
    oWorkPlane = oCompDef.WorkPlanes.AddByPlaneAndOffset(oCompDef.WorkPlanes.Item(3), 1)

    ' Get the browser node definition associated with the work plane.
    Dim oNativeBrowserNodeDef As NativeBrowserNodeDefinition
    oNativeBrowserNodeDef = oPartDoc.BrowserPanes.GetNativeBrowserNodeDefinition(oWorkPlane)

    ' Get the top browser node of the model pane.
    Dim oTopBrowserNode As BrowserNode
    oTopBrowserNode = oPartDoc.BrowserPanes.ActivePane.TopNode

    ' Get the work plane browser node.
    ' This assumes that only one node references the browser node definition.
    ' An example of multiple nodes referencing a single definition is a shared
    ' sketch. The browser may have multiple nodes that represent the same shared
    ' sketch, but all of them reference the same definition.
    Dim oWorkPlaneNode As BrowserNode
    oWorkPlaneNode = oTopBrowserNode.AllReferencedNodes(oNativeBrowserNodeDef).Item(1)

    ' Get the browser node definition from the browser node.
    oNativeBrowserNodeDef = Nothing
    oNativeBrowserNodeDef = oWorkPlaneNode.BrowserNodeDefinition

    ' Get the work plane from the browser node definition.
    oWorkPlane = Nothing
    oWorkPlane = oNativeBrowserNodeDef.NativeObject

    ' Select the work plane to make sure we have the right object.
    oPartDoc.SelectSet.Select(oWorkPlane)
```

Single selection - simple

Description

The following sample demonstrates getting a single selection from the user.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub GetSingleSelection()
    ' Get a feature selection from the user
    Dim oObject As Object
    Set oObject = ThisApplication.CommandManager.Pick(kPartFeatureFilter, "Pick a feature")

    MsgBox "Picked: " & oObject.Name
End Sub
```

Copy Code

```
    ' Get a feature selection from the user
    Dim oObject As Object
    oObject = ThisApplication.CommandManager.Pick(kPartFeatureFilter, "Pick a feature")

    MsgBox("Picked: " & oObject.Name)
```

Add parallel environment with contextual tabs

Description

The following sample demonstrates the use of parallel environments and contextual ribbon tabs.

Code Samples

- VBA
- iLogic
- C#

Run the following macro. Open an assembly document. Go the the "Environments" tab and click on the "Some Analysis" button.

Copy Code

```
Sub AddParallelEnvironment()
    ' Get the Environments collection
    Dim oEnvironments As Environments
    Set oEnvironments = ThisApplication.UserInterfaceManager.Environments

    ' Create a new environment
    Dim oNewEnv As Environment
    Set oNewEnv = oEnvironments.Add("Some Analysis", "SomeAnalysis")

    ' Get the ribbon associated with the assembly environment
    Dim oAssemblyRibbon As Ribbon
    Set oAssemblyRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Assembly")

    ' Create contextual tabs and panels within them
    Dim oContextualTabOne As RibbonTab
    Set oContextualTabOne = oAssemblyRibbon.RibbonTabs.Add("Some Analysis", "SomeAnalysis", "ClientId123", , , True)

    Dim oPanelOne As RibbonPanel
    Set oPanelOne = oContextualTabOne.RibbonPanels.Add("Panel One", "PanelOne", "ClientId123")

    Dim oDef1 As ButtonDefinition
    Set oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("PartExtrudeCmd")

    Call oPanelOne.CommandControls.AddButton(oDef1, True)

    Dim oPanelTwo As RibbonPanel
    Set oPanelTwo = oContextualTabOne.RibbonPanels.Add("Panel Two", "PanelTwo", "ClientId123")

    Dim oDef2 As ButtonDefinition
    Set oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AssemblyPlaceComponentCmd")

    Call oPanelTwo.CommandControls.AddButton(oDef2, True)

    Dim oContextualTabTwo As RibbonTab
    Set oContextualTabTwo = oAssemblyRibbon.RibbonTabs.Add("Some Analysis Extras", "SomeAnalysisExtras", "ClientId123", , , True)

    Dim oPanelThree As RibbonPanel
    Set oPanelThree = oContextualTabTwo.RibbonPanels.Add("Panel Three", "PanelThree", "ClientId123")
    Call oPanelThree.CommandControls.AddButton(oDef1, True)

    ' Associate the contextual tabs with the newly created environment
    ' The contextual tabs will only be displayed when this environment is active
    Dim strTabs(1) As String
    strTabs(0) = "SomeAnalysis"
    strTabs(1) = "SomeAnalysisExtras"

    oNewEnv.AdditionalVisibleRibbonTabs = strTabs

    ' Make the "SomeAnalysis" tab default for the environment
    oNewEnv.DefaultRibbonTab = "SomeAnalysis"

    ' Get the collection of parallel environments and add the new environment
    Dim oParEnvs As EnvironmentList
    Set oParEnvs = ThisApplication.UserInterfaceManager.ParallelEnvironments

    Call oParEnvs.Add(oNewEnv)
```

```
    ' Make the new parallel environment available only within the assembly environment
    ' A ControlDefinition is automatically created when an environment is added to the
    ' parallel environments list. The internal name of the definition is the same as
    ' the internal name of the environment.
    Dim oParallelEnvButton As ControlDefinition
    Set oParallelEnvButton = ThisApplication.CommandManager.ControlDefinitions.Item("SomeAnalysis")

    Dim oEnv As Environment
    For Each oEnv In oEnvironments
        If Not oEnv.InternalName = "AMxAssemblyEnvironment" Then
            On Error Resume Next
            Call oEnv.DisabledCommandList.Add(oParallelEnvButton)
        End If
    Next
End Sub
```

Run the following macro. Open an assembly document. Go the the "Environments" tab and click on the "Some Analysis" button.

Copy Code

```
    ' Get the Environments collection
    Dim oEnvironments As Environments
    oEnvironments = ThisApplication.UserInterfaceManager.Environments

    ' Create a new environment
    Dim oNewEnv As Environment
    oNewEnv = oEnvironments.Add("Some Analysis", "SomeAnalysis")

    ' Get the ribbon associated with the assembly environment
    Dim oAssemblyRibbon As Ribbon
    oAssemblyRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Assembly")

    ' Create contextual tabs and panels within them
    Dim oContextualTabOne As RibbonTab
    oContextualTabOne = oAssemblyRibbon.RibbonTabs.Add("Some Analysis", "SomeAnalysis", "ClientId123", , , True)

    Dim oPanelOne As RibbonPanel
    oPanelOne = oContextualTabOne.RibbonPanels.Add("Panel One", "PanelOne", "ClientId123")

    Dim oDef1 As ButtonDefinition
    oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("PartExtrudeCmd")

    Call oPanelOne.CommandControls.AddButton(oDef1, True)

    Dim oPanelTwo As RibbonPanel
    oPanelTwo = oContextualTabOne.RibbonPanels.Add("Panel Two", "PanelTwo", "ClientId123")

    Dim oDef2 As ButtonDefinition
    oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AssemblyPlaceComponentCmd")

    Call oPanelTwo.CommandControls.AddButton(oDef2, True)

    Dim oContextualTabTwo As RibbonTab
    oContextualTabTwo = oAssemblyRibbon.RibbonTabs.Add("Some Analysis Extras", "SomeAnalysisExtras", "ClientId123", , , True)

    Dim oPanelThree As RibbonPanel
    oPanelThree = oContextualTabTwo.RibbonPanels.Add("Panel Three", "PanelThree", "ClientId123")
    Call oPanelThree.CommandControls.AddButton(oDef1, True)

    ' Associate the contextual tabs with the newly created environment
    ' The contextual tabs will only be displayed when this environment is active
    Dim strTabs(1) As String
    strTabs(0) = "SomeAnalysis"
    strTabs(1) = "SomeAnalysisExtras"

    oNewEnv.AdditionalVisibleRibbonTabs = strTabs

    ' Make the "SomeAnalysis" tab default for the environment
    oNewEnv.DefaultRibbonTab = "SomeAnalysis"

    ' Get the collection of parallel environments and add the new environment
    Dim oParEnvs As EnvironmentList
    oParEnvs = ThisApplication.UserInterfaceManager.ParallelEnvironments

    Call oParEnvs.Add(oNewEnv)

    ' Make the new parallel environment available only within the assembly environment
    ' A ControlDefinition is automatically created when an environment is added to the
    ' parallel environments list. The internal name of the definition is the same as
    ' the internal name of the environment.
    Dim oParallelEnvButton As ControlDefinition
    oParallelEnvButton = ThisApplication.CommandManager.ControlDefinitions.Item("SomeAnalysis")

    Dim oEnv As Environment
    For Each oEnv In oEnvironments
        If Not oEnv.InternalName = "AMxAssemblyEnvironment" Then
            On Error Resume Next
            Call oEnv.DisabledCommandList.Add(oParallelEnvButton)
        End If
    Next
```

Run the following macro. Open an assembly document. Go the the "Environments" tab and click on the "Some Analysis" button. The first line of this sample sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void AddParallelEnvironment()
{
    Application oApp = ThisApplication;

    // Get the Environments collection
    Environments oEnvironments = oApp.UserInterfaceManager.Environments;

    // Create a new environment
    Inventor.Environment oNewEnv = oEnvironments.Add("Some Analysis", "SomeAnalysis", null, null, null);
```

```
    // Get the ribbon associated with the assembly environment
    Ribbon oAssemblyRibbon = oApp.UserInterfaceManager.Ribbons["Assembly"];

    // Create contextual tabs and panels within them
    RibbonTab oContextualTabOne = oAssemblyRibbon.RibbonTabs.Add("Some Analysis", "SomeAnalysis", "ClientId123","" ,false , true);

    RibbonPanel oPanelOne = oContextualTabOne.RibbonPanels.Add("Panel One", "PanelOne", "ClientId123","", false);
    ButtonDefinition oDef1 = (ButtonDefinition)oApp.CommandManager.ControlDefinitions["PartExtrudeCmd"];
    oPanelOne.CommandControls.AddButton(oDef1, true, true, "", false);

    RibbonPanel oPanelTwo = oContextualTabOne.RibbonPanels.Add("Panel Two", "PanelTwo", "ClientId123", "", false);
    ButtonDefinition oDef2 = (ButtonDefinition)oApp.CommandManager.ControlDefinitions["AssemblyPlaceComponentCmd"];
    oPanelTwo.CommandControls.AddButton(oDef2, true, true, "", false);

    RibbonTab oContextualTabTwo = oAssemblyRibbon.RibbonTabs.Add("Some Analysis Extras", "SomeAnalysisExtras", "ClientId123","" ,false

    RibbonPanel oPanelThree = oContextualTabTwo.RibbonPanels.Add("Panel Three", "PanelThree", "ClientId123", "", false);
    oPanelThree.CommandControls.AddButton(oDef1, true, true, "", false);

    // Associate the contextual tabs with the newly created environment
    // The contextual tabs will only be displayed when this environment is active
    String[] strTabs = new String[2];
    strTabs[0] = "SomeAnalysis";
    strTabs[1] = "SomeAnalysisExtras";

    oNewEnv.AdditionalVisibleRibbonTabs = strTabs;

    // Make the "SomeAnalysis" tab default for the environment
    oNewEnv.DefaultRibbonTab = "SomeAnalysis";

    // Get the collection of parallel environments and add the new environment
    EnvironmentList oParEnvs = oApp.UserInterfaceManager.ParallelEnvironments;

    oParEnvs.Add(oNewEnv);

    // Make the new parallel environment available only within the assembly environment
    // A ControlDefinition is automatically created when an environment is added to the
    // parallel environments list. The internal name of the definition is the same as
    // the internal name of the environment.
    ControlDefinition oParallelEnvButton = oApp.CommandManager.ControlDefinitions["SomeAnalysis"];

    int iEnvCount = oEnvironments.Count;
    Inventor.Environment oEnv;
    int i;
    for (i = 1; i <= iEnvCount; i++)
    {
        oEnv = oEnvironments[i];
        if (oEnv.InternalName != "AMxAssemblyEnvironment")
        {
            oEnv.DisabledCommandList.Add(oParallelEnvButton);
        }
    }
}
```

Dockable window

Description

This sample demonstrates creating a dockable window and adding a dialog into it.

Code Samples

- VBA
- iLogic

You need to create the (modeless) dialog and set the hwnd of the dialog in the code below.

Copy Code

```
Sub DockableWindow()
    Dim oUserInterfaceMgr As UserInterfaceManager
    Set oUserInterfaceMgr = ThisApplication.UserInterfaceManager

    ' Create a new dockable window
    Dim oWindow As DockableWindow
    Set oWindow = oUserInterfaceMgr.DockableWindows.Add("SampleClientId", "TestWindowInternalName", "Test Window")

    ' Get the hwnd of the dialog to be added as a child
    ' CHANGE THIS VALUE!
    Dim hwnd As Long
    hwnd = 4851096

    ' Add the dialog as a child to the dockable window
    Call oWindow.AddChild(hwnd)

    ' Don't allow docking to top and bottom
    oWindow.DisabledDockingStates = kDockTop + kDockBottom

    ' Make the window visible
    oWindow.Visible = True
End Sub
```

You need to create the (modeless) dialog and set the hwnd of the dialog in the code below.

Copy Code

```
    Dim oUserInterfaceMgr As UserInterfaceManager
    oUserInterfaceMgr = ThisApplication.UserInterfaceManager
```

```
    ' Create a new dockable window
    Dim oWindow As DockableWindow
    oWindow = oUserInterfaceMgr.DockableWindows.Add("SampleClientId", "TestWindowInternalName", "Test Window")

    ' Get the hwnd of the dialog to be added as a child
    ' CHANGE THIS VALUE!
    Dim hwnd As Long
    hwnd = 4851096

    ' Add the dialog as a child to the dockable window
    Call oWindow.AddChild(hwnd)

    ' Don't allow docking to top and bottom
    oWindow.DisabledDockingStates = kDockTop + kDockBottom

    ' Make the window visible
    oWindow.Visible = True
```

Dock browser pane to a custom ViewFrame

Description

This sample demonstrates how to dock the browser pane to a custom ViewFrame.

Code Samples

- VBA
- iLogic

This sample demonstrates how to dock the browser pane to a custom ViewFrame.

Copy Code

```
Sub DockBrowserPaneToCustomViewFrameSample()
    ' This sample demonstrates how to create a custom ViewFrame and dock the browser pane into it.

    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' This View of the new document is located in the default ViewFrame.
    Dim oView1 As View
    Set oView1 = oDoc.Views(1)

    Dim oViewTab1 As ViewTab
    Set oViewTab1 = oView1.ViewTab

    ' Create a new View for the same document.
    Dim oView2 As View
    Set oView2 = oDoc.Views.Add

    Dim oViewTab2 As ViewTab
    Set oViewTab2 = oView2.ViewTab

    ' Move the second View to generate a new custom ViewFrame, this will also activate the new ViewFrame.
    Dim oViewFrame1 As ViewFrame
    Set oViewFrame1 = oViewTab2.MoveToNewViewFrame(500, 600, 200, 100)

    Dim oBrowserPane As BrowserPane
    Set oBrowserPane = oDoc.BrowserPanes("PmDefault")

    oBrowserPane.SetDockingState kDockLeft, oViewFrame1

End Sub
```

This sample demonstrates how to dock the browser pane to a custom ViewFrame.

Copy Code

```
    ' This sample demonstrates how to create a custom ViewFrame and dock the browser pane into it.

    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    ' This View of the new document is located in the default ViewFrame.
    Dim oView1 As View
    oView1 = oDoc.Views(1)

    Dim oViewTab1 As ViewTab
    oViewTab1 = oView1.ViewTab

    ' Create a new View for the same document.
    Dim oView2 As View
    oView2 = oDoc.Views.Add

    Dim oViewTab2 As ViewTab
    oViewTab2 = oView2.ViewTab

    ' Move the second View to generate a new custom ViewFrame, this will also activate the new ViewFrame.
    Dim oViewFrame1 As ViewFrame
    oViewFrame1 = oViewTab2.MoveToNewViewFrame(500, 600, 200, 100)

    Dim oBrowserPane As BrowserPane
    oBrowserPane = oDoc.BrowserPanes("PmDefault")

    oBrowserPane.SetDockingState(kDockLeft, oViewFrame1)
```

Print list of all Inventor Commands

Description

This sample prints the internal names and descriptions of all commands (aka ControlDefinitions) in Inventor.

Code Samples

- VBA
- iLogic

The sample prints the names and descriptions to a text file (CommandNames.txt) created in C:\Temp directory. You should either have the C:\Temp directory or change the path in the sample.

Copy Code

```
Sub PrintCommandNames()
    Dim oControlDefs As ControlDefinitions
    Set oControlDefs = ThisApplication.CommandManager.ControlDefinitions

    Dim oControlDef As ControlDefinition

    Open "C:\temp\CommandNames.txt" For Output As #1

    Print #1, "Command Name"; Space(49); "Description"; vbNewLine

    For Each oControlDef In oControlDefs
        If Len(Trim(oControlDef.InternalName)) < 60 Then
            sName = Trim(oControlDef.InternalName) & Space(60 - Len(Trim(oControlDef.InternalName)))
        Else
            sName = Trim(oControlDef.InternalName)
        End If

        Print #1, sName; " "; Replace(Trim(oControlDef.DescriptionText), vbCr, "")
    Next
    Close #1
End Sub
```

The sample prints the names and descriptions to a text file (CommandNames.txt) created in C:\Temp directory. You should either have the C:\Temp directory or change the path in the sample.

Copy Code

```
    Imports System.IO

    Dim oControlDefs As ControlDefinitions
    oControlDefs = ThisApplication.CommandManager.ControlDefinitions

    Dim oControlDef As ControlDefinition
    Dim oWriter As System.IO.StreamWriter
        oWriter = New StreamWriter("C:\Temp\CommandNames.txt")

    oWriter.WriteLine("Command Name".PadRight(60) & " Description")
    oWriter.WriteLine()

    For Each oControlDef In oControlDefs
        oWriter.WriteLine(Trim(oControlDef.InternalName).PadRight(60) &" " & Replace(Trim(oControlDef.DescriptionText),vbCr,""))
    Next
    oWriter.Close()
```

Print information about all available ribbons.

Description

This sample prints out all of the elements of the ribbons. This output is very useful when customizing the ribbon to be able to get the names of the various existing ribbons, tabs, and panels.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub PrintRibbon()
    Open "C:\temp\RibbonNames.txt" For Output As #1

    Print #1, "File Controls (Application Menu)"
    Call PrintControls(ThisApplication.UserInterfaceManager.FileBrowserControls, "", 1)
    Print #1, "----------------------------------------------------------------------"

    Print #1, "Help Controls"
    Call PrintControls(ThisApplication.UserInterfaceManager.HelpControls, "", 1)
    Print #1, "----------------------------------------------------------------------"

    Dim oRibbon As Ribbon
    For Each oRibbon In ThisApplication.UserInterfaceManager.Ribbons
        Print #1, "Ribbon: " & oRibbon.InternalName

        Print #1, "    QAT controls"
        Call PrintControls(oRibbon.QuickAccessControls, "            ", 0)
```

```
        Dim oTab As RibbonTab
        For Each oTab In oRibbon.RibbonTabs
            Print #1, "    Tab: " & oTab.DisplayName & ", " & oTab.InternalName & ", Visible: " & oTab.Visible

            Dim oPanel As RibbonPanel
            For Each oPanel In oTab.RibbonPanels
                Print #1, "        Panel: " & oPanel.DisplayName & ", " & oPanel.InternalName & ", Visible: " & oPanel.Visible

                Call PrintControls(oPanel.CommandControls, "            ", 0)

                If oPanel.SlideoutControls.Count > 0 Then
                    Print #1, "            --- Slideout Controls ---"
                    Call PrintControls(oPanel.SlideoutControls, "            ", 0)
                End If
            Next
        Next

    Print #1, "----------------------------------------------------------------------"
    Next
    On Error GoTo 0

    Close #1

    MsgBox "Result written to: C:\temp\RibbonNames.txt"
End Sub

Private Sub PrintControls(Controls As CommandControls, LeadingSpace As String, Level As Integer)
    Dim oControl As CommandControl
    For Each oControl In Controls
        If oControl.ControlType = kSeparatorControl Then
            Print #1, LeadingSpace & Space(Level * 4) & "Control: Seperator"
        Else
            Print #1, LeadingSpace & Space(Level * 4) & "Control: " & oControl.DisplayName & ", " & oControl.InternalName & ", Visible

            If Not oControl.ChildControls Is Nothing Then
                Call PrintControls(oControl.ChildControls, LeadingSpace, Level + 1)
            End If
        End If
    Next
End Sub
```

Copy Code

```
Imports System.IO
Class Test
        Private oWriter As System.IO.StreamWriter
        Sub Main

                oWriter = New StreamWriter("C:\Temp\RibbonNames.txt")

            oWriter.WriteLine("File Controls (Application Menu)")
            Call PrintControls(ThisApplication.UserInterfaceManager.FileBrowserControls, "", 1)
            oWriter.WriteLine("----------------------------------------------------------------------")

            oWriter.WriteLine("Help Controls")
            Call PrintControls(ThisApplication.UserInterfaceManager.HelpControls, "", 1)
            oWriter.WriteLine("----------------------------------------------------------------------")

            Dim oRibbon As Ribbon
            For Each oRibbon In ThisApplication.UserInterfaceManager.Ribbons
                oWriter.WriteLine("Ribbon: " & oRibbon.InternalName)

                oWriter.WriteLine("    QAT controls")
                Call PrintControls(oRibbon.QuickAccessControls, "            ", 0)

                Dim oTab As RibbonTab
                For Each oTab In oRibbon.RibbonTabs
                    oWriter.WriteLine("    Tab: " & oTab.DisplayName & ", " & oTab.InternalName & ", Visible: " & oTab.Visible)

                    Dim oPanel As RibbonPanel
                    For Each oPanel In oTab.RibbonPanels
                        oWriter.WriteLine("        Panel: " & oPanel.DisplayName & ", " & oPanel.InternalName & ", Visible: " & oPanel

                        Call PrintControls(oPanel.CommandControls, "            ", 0)

                        If oPanel.SlideoutControls.Count > 0 Then
                            oWriter.WriteLine("            --- Slideout Controls ---")
                            Call PrintControls(oPanel.SlideoutControls, "            ", 0)
                        End If
                    Next
                Next

            oWriter.WriteLine("----------------------------------------------------------------------")
            Next
            On Error GoTo 0

            oWriter.Close()

            MsgBox("Result written to: C:\temp\RibbonNames.txt")
        End Sub

        Private Sub PrintControls(Controls As CommandControls, LeadingSpace As String, Level As Integer)
            Dim oControl As CommandControl
            For Each oControl In Controls
                If oControl.ControlType = kSeparatorControl Then
                    oWriter.WriteLine(LeadingSpace & Space(Level * 4) & "Control: Seperator")
                Else
                    oWriter.WriteLine(LeadingSpace & Space(Level * 4) & "Control: " & oControl.DisplayName & ", " & oControl.InternalN

                    If Not oControl.ChildControls Is Nothing Then
                        Call PrintControls(oControl.ChildControls, LeadingSpace, Level + 1)
                    End If
                End If
            Next
        End Sub
End Class
```

Creation of an override environment for a document

Description

A new ribbon tab is created and associated with the override environment.

Code Samples

- VBA
- iLogic

Open a part document and run the sample.

Copy Code

```
Sub AddOverrideEnvironment()
    ' Make sure a part document is active
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.ActiveDocument

    Dim oEnvironments As Environments
    Set oEnvironments = ThisApplication.UserInterfaceManager.Environments

    ' Create a new environment
    Dim oOverrideEnv As Environment
    Set oOverrideEnv = oEnvironments.Add("Override", "OverrideEnvironment")

    ' Get the part ribbon
    Dim oPartRibbon As Ribbon
    Set oPartRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Part")

    ' Create a contextual tab to be used as the default for the override environment
    Dim oTabOne As RibbonTab
    Set oTabOne = oPartRibbon.RibbonTabs.Add("Tab One", "TabOne", "ClientId123", "id_TabSheetMetal", True, False)

    ' Create panels with the tab
    Dim oPanelOne As RibbonPanel
    Set oPanelOne = oTabOne.RibbonPanels.Add("Panel One", "PanelOne", "ClientId123")

    Dim oDef1 As ButtonDefinition
    Set oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("PartExtrudeCmd")

    Call oPanelOne.CommandControls.AddButton(oDef1, True)

    Dim oPanelTwo As RibbonPanel
    Set oPanelTwo = oTabOne.RibbonPanels.Add("Panel Two", "PanelTwo", "ClientId123")

    Dim oDef2 As ButtonDefinition
    Set oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("PartRevolveCmd")

    Call oPanelTwo.CommandControls.AddButton(oDef2, True)

    Dim strTabs(0) As String
    strTabs(0) = "TabOne"

    oOverrideEnv.InheritAllRibbonTabs = False
    oOverrideEnv.AdditionalVisibleRibbonTabs = strTabs
    oOverrideEnv.DefaultRibbonTab = "TabOne"

    ' Set the override environment on the active part
    oPartDoc.EnvironmentManager.OverrideEnvironment = oOverrideEnv

End Sub
```

Open a part document and run the sample.

Copy Code

```
    ' Make sure a part document is active
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    Dim oEnvironments As Environments
    oEnvironments = ThisApplication.UserInterfaceManager.Environments

    ' Create a new environment
    Dim oOverrideEnv As Environment
    oOverrideEnv = oEnvironments.Add("Override", "OverrideEnvironment")

    ' Get the part ribbon
    Dim oPartRibbon As Ribbon
    oPartRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Part")

    ' Create a contextual tab to be used as the default for the override environment
    Dim oTabOne As RibbonTab
    oTabOne = oPartRibbon.RibbonTabs.Add("Tab One", "TabOne", "ClientId123", "id_TabSheetMetal", True, False)

    ' Create panels with the tab
    Dim oPanelOne As RibbonPanel
    oPanelOne = oTabOne.RibbonPanels.Add("Panel One", "PanelOne", "ClientId123")

    Dim oDef1 As ButtonDefinition
    oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("PartExtrudeCmd")

    Call oPanelOne.CommandControls.AddButton(oDef1, True)

    Dim oPanelTwo As RibbonPanel
    oPanelTwo = oTabOne.RibbonPanels.Add("Panel Two", "PanelTwo", "ClientId123")
```

```
    Dim oDef2 As ButtonDefinition
    oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("PartRevolveCmd")

    Call oPanelTwo.CommandControls.AddButton(oDef2, True)

    Dim strTabs(0) As String
    strTabs(0) = "TabOne"

    oOverrideEnv.InheritAllRibbonTabs = False
    oOverrideEnv.AdditionalVisibleRibbonTabs = strTabs
    oOverrideEnv.DefaultRibbonTab = "TabOne"

    ' Set the override environment on the active part
    oPartDoc.EnvironmentManager.OverrideEnvironment = oOverrideEnv
```

Using Inventor's error dialog

Description

Demonstrates using Inventor's error dialog.

Code Samples

- VBA
- iLogic

Copy Code

```
Public Sub ErrorDialog()
    ' Create a new part document
    Dim oPartDoc As PartDocument
    Set oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    Set oCompDef = oPartDoc.ComponentDefinition

    Dim oSketch As PlanarSketch
    Set oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    Dim oTransGeom As TransientGeometry
    Set oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    Set oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    Set oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    Set oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    Set oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oErrorMgr As ErrorManager
    Set oErrorMgr = ThisApplication.ErrorManager

    Dim oMessageSection As MessageSection
    Set oMessageSection = oErrorMgr.StartMessageSection

    On Error Resume Next

    Dim oTransaction As Transaction
    Set oTransaction = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Edit Feature")

    ' Delete all sketch lines making the sketch profile invalid.
    Dim oSketchLine As SketchLine
    For Each oSketchLine In oRectangleLines
        oSketchLine.Delete
    Next

    oPartDoc.Update

    If oMessageSection.HasErrors Or oMessageSection.HasWarnings Then
        ' End section by adopting all messages in section
        Call oMessageSection.AdoptMessages("Feature edit failed", True)

        ' Display the messages
        Dim kButtonType As ButtonTypeEnum
        kButtonType = oErrorMgr.Show("Sample of Feature Edit Error", True, False)

        If kButtonType = kAcceptButtonType Then
            ' Go through with the invalid edit if user accepts
            oTransaction.End
        Else
            oTransaction.Abort
        End If
    Else
        ' End section by clearing all messages in section
        oMessageSection.ClearMessages
        oTransaction.End
```

```
    End If
End Sub
```

Copy Code

```
    ' Create a new part document
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oCompDef As PartComponentDefinition
    oCompDef = oPartDoc.ComponentDefinition

    Dim oSketch As PlanarSketch
    oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes(3))

    Dim oTransGeom As TransientGeometry
    oTransGeom = ThisApplication.TransientGeometry

    ' Draw a 4cm x 3cm rectangle with the corner at (0,0)
    Dim oRectangleLines As SketchEntitiesEnumerator
    oRectangleLines = oSketch.SketchLines.AddAsTwoPointRectangle( _
                              oTransGeom.CreatePoint2d(0, 0), _
                              oTransGeom.CreatePoint2d(4, 3))

    ' Create a profile.
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid

    ' Create a base extrusion 1cm thick.
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oCompDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, kNewBodyOperation)
    Call oExtrudeDef.SetDistanceExtent(1, kNegativeExtentDirection)
    Dim oExtrude As ExtrudeFeature
    oExtrude = oCompDef.Features.ExtrudeFeatures.Add(oExtrudeDef)

    Dim oErrorMgr As ErrorManager
    oErrorMgr = ThisApplication.ErrorManager

    Dim oMessageSection As MessageSection
    oMessageSection = oErrorMgr.StartMessageSection

    On Error Resume Next

    Dim oTransaction As Transaction
    oTransaction = ThisApplication.TransactionManager.StartTransaction(oPartDoc, "Edit Feature")

    ' Delete all sketch lines making the sketch profile invalid.
    Dim oSketchLine As SketchLine
    For Each oSketchLine In oRectangleLines
        oSketchLine.Delete
    Next

    oPartDoc.Update

    If oMessageSection.HasErrors Or oMessageSection.HasWarnings Then
        ' End section by adopting all messages in section
        Call oMessageSection.AdoptMessages("Feature edit failed", True)

        ' Temporarily enable dialogs.
        ThisApplication.SilentOperation = False

        ' Display the messages
        Dim kButtonType As ButtonTypeEnum
        kButtonType = oErrorMgr.Show("Sample of Feature Edit Error", True, False)

        If kButtonType = kAcceptButtonType Then
            ' Go through with the invalid edit if user accepts
            oTransaction.End
        Else
            oTransaction.Abort
        End If
    Else
        ' End section by clearing all messages in section
        oMessageSection.ClearMessages
        oTransaction.End
    End If
```

Display custom error messages

Description

Demonstrates displaying custom error messages.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub DisplayErrorsInInventorDialog()
    ' Set a reference to the ErrorManager object
    Dim oErrorMgr As ErrorManager
    Set oErrorMgr = ThisApplication.ErrorManager

    ' Start a message section
    Dim oMsgSection1 As MessageSection
```

```
    Set oMsgSection1 = oErrorMgr.StartMessageSection

    ' Start another (nested) message section
    Dim oMsgSection2 As MessageSection
    Set oMsgSection2 = oErrorMgr.StartMessageSection

    ' Add a message
    Call oErrorMgr.AddMessage("My third level error", True)

    ' Adopt the above message under a new message
    Call oMsgSection2.AdoptMessages("My second level error", True)

    ' Adopt the above messages under the top level message
    Call oMsgSection1.AdoptMessages("My first level error", True)

    ' Show the error dialog
    Call oErrorMgr.Show("My errors", False, False)
End Sub
```

Copy Code

```
    ' Set a reference to the ErrorManager object
    Dim oErrorMgr As ErrorManager
    oErrorMgr = ThisApplication.ErrorManager

    ' Start a message section
    Dim oMsgSection1 As MessageSection
    oMsgSection1 = oErrorMgr.StartMessageSection

    ' Start another (nested) message section
    Dim oMsgSection2 As MessageSection
    oMsgSection2 = oErrorMgr.StartMessageSection

    ' Add a message
    Call oErrorMgr.AddMessage("My third level error", True)

    ' Adopt the above message under a new message
    Call oMsgSection2.AdoptMessages("My second level error", True)

    ' Adopt the above messages under the top level message
    Call oMsgSection1.AdoptMessages("My first level error", True)

    ' Temporarily enable dialogs.
     ThisApplication.SilentOperation = False

    ' Show the error dialog
    Call oErrorMgr.Show("My errors", False, False)
```

File Dialog

Description

This sample demonstrates the use of the FileDialog object. The only requirement to run this sample is to have Inventor open.

Code Samples

- VBA
- iLogic
- C#

Copy Code

```
Public Sub TestFileDialog()
    ' Create a new FileDialog object.
    Dim oFileDlg As FileDialog
    Call ThisApplication.CreateFileDialog(oFileDlg)

    ' Define the filter to select part and assembly files or any file.
    oFileDlg.Filter = "Inventor Files (*.iam;*.ipt)|*.iam;*.ipt|All Files (*.*)|*.*"

    ' Define the part and assembly files filter to be the default filter.
    oFileDlg.FilterIndex = 1

    ' Set the title for the dialog.
    oFileDlg.DialogTitle = "Open File Test"

    ' Set the initial directory that will be displayed in the dialog.
    oFileDlg.InitialDirectory = "C:\Temp"

    ' Set the flag so an error will be raised if the user clicks the Cancel button.
    oFileDlg.CancelError = True

    ' Show the open dialog.  The same procedure is also used for the Save dialog.
    ' The commented code can be used for the Save dialog.
    On Error Resume Next
    oFileDlg.ShowOpen
'    oFileDlg.ShowSave

    ' If an error was raised, the user clicked cancel, otherwise display the filename.
    If Err Then
        MsgBox "User cancelled out of dialog"
    ElseIf oFileDlg.FileName <> "" Then
        MsgBox "File " & oFileDlg.FileName & " was selected."
    End If
End Sub
```

Copy Code

```
    ' Create a new FileDialog object.
    Dim oFileDlg As FileDialog
    Call ThisApplication.CreateFileDialog(oFileDlg)

    ' Define the filter to select part and assembly files or any file.
    oFileDlg.Filter = "Inventor Files (*.iam;*.ipt)|*.iam;*.ipt|All Files (*.*)|*.*"

    ' Define the part and assembly files filter to be the default filter.
    oFileDlg.FilterIndex = 1

    ' Set the title for the dialog.
    oFileDlg.DialogTitle = "Open File Test"

    ' Set the initial directory that will be displayed in the dialog.
    oFileDlg.InitialDirectory = "C:\Temp"

    ' Set the flag so an error will be raised if the user clicks the Cancel button.
    oFileDlg.CancelError = True

    ' Show the open dialog.  The same procedure is also used for the Save dialog.
    ' The commented code can be used for the Save dialog.
    On Error Resume Next
    oFileDlg.ShowOpen
'    oFileDlg.ShowSave

    ' If an error was raised, the user clicked cancel, otherwise display the filename.
    If Err.Number > 0  Then
        MsgBox("User cancelled out of dialog")
    ElseIf oFileDlg.FileName <> "" Then
        MsgBox("File " & oFileDlg.FileName & " was selected.")
    End If
```

The first line sets the oApp variable to ThisApplication - this should be appropriately changed.

Copy Code

```
public void TestFileDialog()
{
    Application oApp = ThisApplication;

    // Create a new FileDialog object.
    FileDialog oFileDlg;
    oApp.CreateFileDialog(out(oFileDlg));

    // Define the filter to select part and assembly files or any file.
    oFileDlg.Filter = "Inventor Files (*.iam;*.ipt)|*.iam;*.ipt|All Files (*.*)|*.*";

    // Define the part and assembly files filter to be the default filter.
    oFileDlg.FilterIndex = 1;

    // Set the title for the dialog.
    oFileDlg.DialogTitle = "Open File Test";

    // Set the initial directory that will be displayed in the dialog.
    oFileDlg.InitialDirectory = "C:/Temp";

    // Set the flag so an error will not be raised if the user clicks the Cancel button.
    oFileDlg.CancelError = false;

    // Show the open dialog.  The same procedure is also used for the Save dialog.
    // The commented code can be used for the Save dialog.
    oFileDlg.ShowOpen();
    // oFileDlg.ShowSave();

    System.Windows.Forms.MessageBox.Show("File " + oFileDlg.FileName + " was selected.", "Selected file");
}
```

MiniToolbarsample

Description

This sample demonstrates how to create sketch slot with minitoolbar.

Code Samples

- VBA
- iLogic

Have a sektch in edit mode before running below sample.

Copy Code

```
' This sample demonstrates creating sketch slot using mini-toolbar
Sub CreateSketchSlotSample()
    Dim oActiveEnv As Environment
    Set oActiveEnv = ThisApplication.UserInterfaceManager.ActiveEnvironment

    If oActiveEnv.InternalName <> "PMxPartSketchEnvironment" And _
        oActiveEnv.InternalName <> "AMxAssemblySketchEnvironment" And _
        oActiveEnv.InternalName <> "DLxDrawingSketchEnvironment" Then
        MsgBox "Please activate a sketch environment first!"
        Exit Sub
    End If

    Dim oMiniToolbar As MiniToolbar
    Set oMiniToolbar = ThisApplication.CommandManager.CreateMiniToolbar
```

```
    oMiniToolbar.ShowOK = True
    oMiniToolbar.ShowApply = True
    oMiniToolbar.ShowCancel = True

    Dim oControls As MiniToolbarControls
    Set oControls = oMiniToolbar.Controls
    oControls.Item("MTB_Options").Visible = False

    Dim oDescriptionLabel As MiniToolbarControl
    Set oDescriptionLabel = oControls.AddLabel("Description", "This toolbar is to create sketch slot:", "MiniToolbar sample to show ho
    oControls.AddNewLine

    ' Define the first center position.
    Dim oEndCenterOne As MiniToolbarButton
    Set oEndCenterOne = oControls.AddButton("FirstCenter: ", "First Center:     ", "Specify the first center of sketch slot")


    Dim oEndCenterOneX As MiniToolbarValueEditor
    Set oEndCenterOneX = oControls.AddValueEditor("FirstCenterX", "", kLengthUnits, "", "X:")
    oEndCenterOneX.Expression = "0"
    oEndCenterOneX.SetFocus

    Dim oEndCenterOneY As MiniToolbarValueEditor
    Set oEndCenterOneY = oControls.AddValueEditor("FirstCenterY", "", kLengthUnits, "", "Y:")
    oEndCenterOneY.Expression = "0"
    oControls.AddNewLine

    ' Define the second center position.
    Dim oEndCenterTwo As MiniToolbarButton
    Set oEndCenterTwo = oControls.AddButton("SecondCenter:", "Second Center:", "Specify the second center of sketch slot")

    Dim oEndCenterTwoX As MiniToolbarValueEditor
    Set oEndCenterTwoX = oControls.AddValueEditor("SecondCenterX", "", kLengthUnits, "", "X:")
    oEndCenterTwoX.Expression = "3"

    Dim oEndCenterTwoY As MiniToolbarValueEditor
    Set oEndCenterTwoY = oControls.AddValueEditor("SecondCenterY", "", kLengthUnits, "", "Y:")
    oEndCenterTwoY.Expression = "0"

    oControls.AddNewLine

    ' Define the width of sketch slot.
    Dim oWidthValue As MiniToolbarValueEditor
    Set oWidthValue = oControls.AddValueEditor("WidthValue", "", kLengthUnits, "", "Width:")
    oWidthValue.Expression = "1"

    ' Define if display the center line of sketch slot.
    Dim oDisplayCenterline As MiniToolbarCheckBox
    Set oDisplayCenterline = oControls.AddCheckBox("DisplayCenterline", "Display center line", "Check this to display center line of s

    ' Set the position of mini-toolbar
    Dim oPosition As Point2d
    Set oPosition = ThisApplication.TransientGeometry.CreatePoint2d(ThisApplication.ActiveView.Left, ThisApplication.ActiveView.Top)
    oMiniToolbar.Position = oPosition

    oMiniToolbar.Visible = True

    Dim oMiniToolbarEvents As New clsMiniToolbarEvents

    Call oMiniToolbarEvents.Init(oMiniToolbar)

End Sub


'************************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsMiniToolbarEvents". The name can be
' changed but you'll need to change the declaration in the
' calling function "CreateSketchSlotSample" to use the new name.
Private WithEvents m_EndCenterOneX As MiniToolbarValueEditor
Private WithEvents m_EndCenterOneY As MiniToolbarValueEditor
Private WithEvents m_EndCenterSecondX As MiniToolbarValueEditor
Private WithEvents m_EndCenterSecondY As MiniToolbarValueEditor
Private WithEvents m_Width As MiniToolbarValueEditor
Private WithEvents m_MiniToolbar As MiniToolbar
Private oDisplayCenterline As MiniToolbarCheckBox
Private m_Sketch As Sketch
Private bCenterline As Boolean
Private bStop As Boolean

Public Sub Init(oMiniToolbar As MiniToolbar)
    Set m_MiniToolbar = oMiniToolbar


    Set m_EndCenterOneX = m_MiniToolbar.Controls.Item("FirstCenterX")
    Set m_EndCenterOneY = m_MiniToolbar.Controls.Item("FirstCenterY")
    Set m_EndCenterSecondX = m_MiniToolbar.Controls.Item("SecondCenterX")
    Set m_EndCenterSecondY = m_MiniToolbar.Controls.Item("SecondCenterY")
    Set m_Width = m_MiniToolbar.Controls.Item("WidthValue")

    Set oDisplayCenterline = m_MiniToolbar.Controls.Item("DisplayCenterline")


    Set m_Sketch = ThisApplication.ActiveEditObject
    bStop = False

    Do
        ThisApplication.UserInterfaceManager.DoEvents

    Loop Until bStop

End Sub


Private Sub m_MiniToolbar_OnApply()
    CreateSlot
```

```
End Sub

Private Sub m_MiniToolbar_OnCancel()
    bStop = True

End Sub


Private Sub m_MiniToolbar_OnOK()
    bStop = True

    CreateSlot

End Sub

Private Sub CreateSlot()
    If Not (m_EndCenterOneX.IsExpressionValid And m_EndCenterOneY.IsExpressionValid And m_EndCenterSecondX.IsExpressionValid And m_End
        MsgBox "Invalid values for end center positions!"
        Exit Sub
    End If

    bCenterline = oDisplayCenterline.Checked

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    Dim oEndCenterOne As Point2d
    Dim oEndCenterTwo As Point2d

    ' Start transaction for creating slot.
    Dim oTransaction As Transaction
    Set oTransaction = ThisApplication.TransactionManager.StartTransaction(ThisApplication.ActiveDocument, "Create slot")

        ' If the two centers are vertical
        If Abs(m_EndCenterOneX.Value - m_EndCenterSecondX.Value) < 0.000001 Then
            If (m_EndCenterOneY.Value > m_EndCenterSecondY.Value) Then
                Set oEndCenterOne = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
                Set oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
            Else
                Set oEndCenterOne = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
                Set oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
            End If

            If oEndCenterOne.IsEqualTo(oEndCenterTwo, 0.000001) Then
                MsgBox "The two centers are coincident!"
                Exit Sub
            End If

            ' Create the top arc
            Set oEndArcOne = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterOne, oTG.CreatePoint2d(oEndCenterOne.X + 0.1, oEnd
            ' Create the bottom arc
            Set oEndArcTwo = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterTwo, oTG.CreatePoint2d(oEndCenterTwo.X - 0.1, oEnd
        'If the two centers are not vertical
        Else
            If m_EndCenterOneX.Value < m_EndCenterSecondX.Value Then
                Set oEndCenterOne = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
                Set oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
            ElseIf m_EndCenterOneX.Value > m_EndCenterSecondX.Value Then
                Set oEndCenterOne = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
                Set oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
            End If

            If oEndCenterOne.IsEqualTo(oEndCenterTwo, 0.000001) Then
                MsgBox "The two centers are coincident!"
                Exit Sub
            End If

            Set oEndArcOne = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterOne, oTG.CreatePoint2d(m_EndCenterOneX.Value, m_En
            Set oEndArcTwo = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterTwo, oTG.CreatePoint2d(m_EndCenterSecondX.Value, m


        End If

        Dim dWidth As Double
        dWidth = m_Width.Value


        ' Create center line if required
        If bCenterline Then
            Dim oCenterline As SketchLine
            Set oCenterline = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.CenterSketchPoint, oEndArcTwo.CenterSketchPoint)

            oCenterline.Construction = True
        End If


        Dim oGround1 As GroundConstraint
        Dim oGround2 As GroundConstraint
        Set oGround1 = m_Sketch.GeometricConstraints.AddGround(oEndArcOne.CenterSketchPoint)
        Set oGround2 = m_Sketch.GeometricConstraints.AddGround(oEndArcTwo.CenterSketchPoint)

        ' Create sketch lines of slot
        Dim oLine1 As SketchLine
        Dim oLine2 As SketchLine
        Set oLine1 = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.StartSketchPoint, oEndArcTwo.EndSketchPoint)
        Set oLine2 = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.EndSketchPoint, oEndArcTwo.StartSketchPoint)

        ' Add geometric constraints to the sketch entities
        Call m_Sketch.GeometricConstraints.AddEqualRadius(oEndArcOne, oEndArcTwo)
        Call m_Sketch.GeometricConstraints.AddTangent(oLine1, oEndArcOne)
        Call m_Sketch.GeometricConstraints.AddTangent(oLine1, oEndArcTwo)
        Call m_Sketch.GeometricConstraints.AddTangent(oLine2, oEndArcOne)
        Call m_Sketch.GeometricConstraints.AddTangent(oLine2, oEndArcTwo)

        ' Add dimensional constraints to the sketch entities
        Dim oDiameter As DiameterDimConstraint
        Set oDiameter = m_Sketch.DimensionConstraints.AddDiameter(oEndArcOne, oEndArcOne.CenterSketchPoint.Geometry)
```

```
        oDiameter.Parameter.Value = dWidth

        ThisApplication.ActiveDocument.Update
        oDiameter.Delete
        oGround1.Delete
        oGround2.Delete

    oTransaction.End

End Sub
```

Have a sektch in edit mode before running below sample.

Copy Code

```
' This sample demonstrates creating sketch slot using mini-toolbar
Sub Main
    Dim oActiveEnv As Environment
    oActiveEnv = ThisApplication.UserInterfaceManager.ActiveEnvironment

    If oActiveEnv.InternalName <> "PMxPartSketchEnvironment" And _
        oActiveEnv.InternalName <> "AMxAssemblySketchEnvironment" And _
        oActiveEnv.InternalName <> "DLxDrawingSketchEnvironment" Then
        MsgBox("Please activate a sketch environment first!")
        Exit Sub
    End If

    Dim oMiniToolbar As MiniToolbar
    oMiniToolbar = ThisApplication.CommandManager.CreateMiniToolbar

    oMiniToolbar.ShowOK = True
    oMiniToolbar.ShowApply = True
    oMiniToolbar.ShowCancel = True

    Dim oControls As MiniToolbarControls
    oControls = oMiniToolbar.Controls
    oControls.Item("MTB_Options").Visible = False

    Dim oDescriptionLabel As MiniToolbarControl
    oDescriptionLabel = oControls.AddLabel("Description", "This toolbar is to create sketch slot:", "MiniToolbar sample to show how to
    oControls.AddNewLine

    ' Define the first center position.
    Dim oEndCenterOne As MiniToolbarButton
    oEndCenterOne = oControls.AddButton("FirstCenter: ", "First Center:     ", "Specify the first center of sketch slot")


    Dim oEndCenterOneX As MiniToolbarValueEditor
    oEndCenterOneX = oControls.AddValueEditor("FirstCenterX", "", kLengthUnits, "", "X:")
    oEndCenterOneX.Expression = "0"
    oEndCenterOneX.SetFocus

    Dim oEndCenterOneY As MiniToolbarValueEditor
    oEndCenterOneY = oControls.AddValueEditor("FirstCenterY", "", kLengthUnits, "", "Y:")
    oEndCenterOneY.Expression = "0"
    oControls.AddNewLine

    ' Define the second center position.
    Dim oEndCenterTwo As MiniToolbarButton
    oEndCenterTwo = oControls.AddButton("SecondCenter:", "Second Center:", "Specify the second center of sketch slot")

    Dim oEndCenterTwoX As MiniToolbarValueEditor
    oEndCenterTwoX = oControls.AddValueEditor("SecondCenterX", "", kLengthUnits, "", "X:")
    oEndCenterTwoX.Expression = "3"

    Dim oEndCenterTwoY As MiniToolbarValueEditor
    oEndCenterTwoY = oControls.AddValueEditor("SecondCenterY", "", kLengthUnits, "", "Y:")
    oEndCenterTwoY.Expression = "0"

    oControls.AddNewLine

    ' Define the width of sketch slot.
    Dim oWidthValue As MiniToolbarValueEditor
    oWidthValue = oControls.AddValueEditor("WidthValue", "", kLengthUnits, "", "Width:")
    oWidthValue.Expression = "1"

    ' Define if display the center line of sketch slot.
    Dim oDisplayCenterline As MiniToolbarCheckBox
    oDisplayCenterline = oControls.AddCheckBox("DisplayCenterline", "Display center line", "Check this to display center line of slot"

    ' Set the position of mini-toolbar
    Dim oPosition As Point2d
    oPosition = ThisApplication.TransientGeometry.CreatePoint2d(ThisApplication.ActiveView.Left, ThisApplication.ActiveView.Top)
    oMiniToolbar.Position = oPosition

    oMiniToolbar.Visible = True

    Dim oMiniToolbarEvents As New clsMiniToolbarEvents

    Call oMiniToolbarEvents.Init(oMiniToolbar, ThisApplication)

End Sub

Class clsMiniToolbarEvents
        Private ThisApplication As Inventor.Application
        Private WithEvents m_EndCenterOneX As MiniToolbarValueEditor
        Private WithEvents m_EndCenterOneY As MiniToolbarValueEditor
        Private WithEvents m_EndCenterSecondX As MiniToolbarValueEditor
        Private WithEvents m_EndCenterSecondY As MiniToolbarValueEditor
        Private WithEvents m_Width As MiniToolbarValueEditor
        Private WithEvents m_MiniToolbar As MiniToolbar
        Private oDisplayCenterline As MiniToolbarCheckBox
        Private m_Sketch As Sketch
        Private bCenterline As Boolean
        Private bStop As Boolean

        Public Sub Init(oMiniToolbar As MiniToolbar, oApp As Inventor.Application)
            ThisApplication = oApp
```

```
        m_MiniToolbar = oMiniToolbar

    m_EndCenterOneX = m_MiniToolbar.Controls.Item("FirstCenterX")
    m_EndCenterOneY = m_MiniToolbar.Controls.Item("FirstCenterY")
    m_EndCenterSecondX = m_MiniToolbar.Controls.Item("SecondCenterX")
    m_EndCenterSecondY = m_MiniToolbar.Controls.Item("SecondCenterY")
    m_Width = m_MiniToolbar.Controls.Item("WidthValue")

    oDisplayCenterline = m_MiniToolbar.Controls.Item("DisplayCenterline")


    m_Sketch = ThisApplication.ActiveEditObject
    bStop = False

    Do
        ThisApplication.UserInterfaceManager.DoEvents

    Loop Until bStop

End Sub


Private Sub m_MiniToolbar_OnApply() Handles m_MiniToolbar.OnApply
    CreateSlot
End Sub

Private Sub m_MiniToolbar_OnCancel() Handles m_MiniToolbar.OnCancel
    bStop = True
End Sub


Private Sub m_MiniToolbar_OnOK() Handles m_MiniToolbar.OnOK
    bStop = True
    CreateSlot()
End Sub

Private Sub CreateSlot()
    If Not (m_EndCenterOneX.IsExpressionValid And m_EndCenterOneY.IsExpressionValid And m_EndCenterSecondX.IsExpressionValid A
        MsgBox("Invalid values for end center positions!")
        Exit Sub
    End If

    bCenterline = oDisplayCenterline.Checked

    Dim oTG As TransientGeometry
    oTG = ThisApplication.TransientGeometry

    Dim oEndCenterOne As Point2d
    Dim oEndCenterTwo As Point2d

    ' Start transaction for creating slot.
    Dim oTransaction As Transaction
    oTransaction = ThisApplication.TransactionManager.StartTransaction(ThisApplication.ActiveDocument, "Create slot")

        ' If the two centers are vertical
        If Abs(m_EndCenterOneX.Value - m_EndCenterSecondX.Value) < 0.000001 Then
            If (m_EndCenterOneY.Value > m_EndCenterSecondY.Value) Then
                oEndCenterOne = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
                oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
            Else
                oEndCenterOne = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
                oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
            End If

            If oEndCenterOne.IsEqualTo(oEndCenterTwo, 0.000001) Then
                MsgBox("The two centers are coincident!")
                Exit Sub
            End If

            ' Create the top arc
            oEndArcOne = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterOne, oTG.CreatePoint2d(oEndCenterOne.X + 0.1, 
            ' Create the bottom arc
            oEndArcTwo = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterTwo, oTG.CreatePoint2d(oEndCenterTwo.X - 0.1, 
        'If the two centers are not vertical
        Else
            If m_EndCenterOneX.Value < m_EndCenterSecondX.Value Then
                oEndCenterOne = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
                oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
            ElseIf m_EndCenterOneX.Value > m_EndCenterSecondX.Value Then
                oEndCenterOne = oTG.CreatePoint2d(m_EndCenterSecondX.Value, m_EndCenterSecondY.Value)
                oEndCenterTwo = oTG.CreatePoint2d(m_EndCenterOneX.Value, m_EndCenterOneY.Value)
            End If

            If oEndCenterOne.IsEqualTo(oEndCenterTwo, 0.000001) Then
                MsgBox("The two centers are coincident!")
                Exit Sub
            End If

            oEndArcOne = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterOne, oTG.CreatePoint2d(m_EndCenterOneX.Value, 
            oEndArcTwo = m_Sketch.SketchArcs.AddByCenterStartEndPoint(oEndCenterTwo, oTG.CreatePoint2d(m_EndCenterSecondX.Valu


        End If

        Dim dWidth As Double
        dWidth = m_Width.Value


        ' Create center line if required
        If bCenterline Then
            Dim oCenterline As SketchLine
            oCenterline = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.CenterSketchPoint, oEndArcTwo.CenterSketchPoint)

            oCenterline.Construction = True
        End If
```

```
                Dim oGround1 As GroundConstraint
                Dim oGround2 As GroundConstraint
                oGround1 = m_Sketch.GeometricConstraints.AddGround(oEndArcOne.CenterSketchPoint)
                oGround2 = m_Sketch.GeometricConstraints.AddGround(oEndArcTwo.CenterSketchPoint)

                ' Create sketch lines of slot
                Dim oLine1 As SketchLine
                Dim oLine2 As SketchLine
                oLine1 = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.StartSketchPoint, oEndArcTwo.EndSketchPoint)
                oLine2 = m_Sketch.SketchLines.AddByTwoPoints(oEndArcOne.EndSketchPoint, oEndArcTwo.StartSketchPoint)

                ' Add geometric constraints to the sketch entities
                Call m_Sketch.GeometricConstraints.AddEqualRadius(oEndArcOne, oEndArcTwo)
                Call m_Sketch.GeometricConstraints.AddTangent(oLine1, oEndArcOne)
                Call m_Sketch.GeometricConstraints.AddTangent(oLine1, oEndArcTwo)
                Call m_Sketch.GeometricConstraints.AddTangent(oLine2, oEndArcOne)
                Call m_Sketch.GeometricConstraints.AddTangent(oLine2, oEndArcTwo)

                ' Add dimensional constraints to the sketch entities
                Dim oDiameter As DiameterDimConstraint
                oDiameter = m_Sketch.DimensionConstraints.AddDiameter(oEndArcOne, oEndArcOne.CenterSketchPoint.Geometry)
                oDiameter.Parameter.Value = dWidth

                ThisApplication.ActiveDocument.Update
                oDiameter.Delete
                oGround1.Delete
                oGround2.Delete

            oTransaction.End

        End Sub
End Class
```

Using Inventor's progress bars

Description

Demonstrates using Inventor's progress bar.

Code Samples

- VBA
- iLogic

Copy Code

```
' 64-bit version
Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As LongLong)

' 32-bit version
'Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Public Sub TestDialogProgressBar()
    Dim iStepCount As Long
    iStepCount = 50

    ' Create a new ProgressBar object.
    Dim oProgressBar As ProgressBar
    Set oProgressBar = ThisApplication.CreateProgressBar(False, iStepCount, "Test Progress")

    ' Set the message for the progress bar
    oProgressBar.Message = "Executing some process"

    Dim i As Long
    For i = 1 To iStepCount
        ' Sleep 0.2 sec to simulate some process
        Sleep 200
        oProgressBar.Message = "Executing some process - " & i
        oProgressBar.UpdateProgress
    Next

    ' Terminate the progress bar.
    oProgressBar.Close
End Sub

Public Sub TestStatusBarProgressBar()
    Dim iStepCount As Long
    iStepCount = 50

    ' Create a new ProgressBar object.
    Dim oProgressBar As ProgressBar
    Set oProgressBar = ThisApplication.CreateProgressBar(True, iStepCount, "Test Progress")

    ' Set the message for the progress bar
    oProgressBar.Message = "Executing some process"

    Dim i As Long
    For i = 1 To iStepCount
        ' Sleep 0.2 sec to simulate some process
        Sleep 200
        oProgressBar.Message = "Executing some process - " & i
        oProgressBar.UpdateProgress
    Next

    ' Terminate the progress bar.
    oProgressBar.Close
End Sub
```

Copy Code

```
Class Test
        Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

        Sub Main
                TestDialogProgressBar()
                TestStatusBarProgressBar()
        End Sub

        Sub TestDialogProgressBar()

            Dim iStepCount As Long
            iStepCount = 50

            ' Create a new ProgressBar object.
            Dim oProgressBar As ProgressBar
            oProgressBar = ThisApplication.CreateProgressBar(False, iStepCount, "Test Progress")

            ' Set the message for the progress bar
            oProgressBar.Message = "Executing some process"

            Dim i As Long
            For i = 1 To iStepCount
                ' Sleep 0.2 sec to simulate some process
                Sleep(200)
                oProgressBar.Message = "Executing some process - " & i
                oProgressBar.UpdateProgress
            Next

            ' Terminate the progress bar.
            oProgressBar.Close
        End Sub

        Public Sub TestStatusBarProgressBar()
            Dim iStepCount As Long
            iStepCount = 50

            ' Create a new ProgressBar object.
            Dim oProgressBar As ProgressBar
            oProgressBar = ThisApplication.CreateProgressBar(True, iStepCount, "Test Progress")

            ' Set the message for the progress bar
            oProgressBar.Message = "Executing some process"

            Dim i As Long
            For i = 1 To iStepCount
                ' Sleep 0.2 sec to simulate some process
                Sleep(200)
                oProgressBar.Message = "Executing some process - " & i
                oProgressBar.UpdateProgress
            Next

            ' Terminate the progress bar.
            oProgressBar.Close
        End Sub
End Class
```

Create a ribbon panel in an existing tab

Description

Demonstrates creating a new ribbon panel within an existing ribbon tab.

Code Samples

- VBA
- iLogic

Copy Code

```
Sub AddPanelToToolsTab()
    ' Get the ribbon associated with the part document
    Dim oPartRibbon As Ribbon
    Set oPartRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Part")

    ' Get the "Tools" tab
    Dim oTab As RibbonTab
    Set oTab = oPartRibbon.RibbonTabs.Item("id_TabTools")

    ' Create a panel named "Update", positioned after the "Measure" panel in the Tools tab.
    Dim oPanel As RibbonPanel
    Set oPanel = oTab.RibbonPanels.Add("Update", "ToolsTabUpdatePanel", "SampleClientId", "id_PanelP_ToolsMeasure")

    ' Get the update commands
    Dim oDef1 As ButtonDefinition
    Set oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("AppLocalUpdateCmd")

    Dim oDef2 As ButtonDefinition
    Set oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AppUpdateMassPropertiesCmd")

    Dim oDefs As ObjectCollection
    Set oDefs = ThisApplication.TransientObjects.CreateObjectCollection

    oDefs.Add oDef1
    oDefs.Add oDef2
```

```
    ' Create a split button control
    Call oPanel.CommandControls.AddSplitButton(oDef1, oDefs, True)

    ' Get the rebuild command
    Dim oDef3 As ButtonDefinition
    Set oDef3 = ThisApplication.CommandManager.ControlDefinitions.Item("AppRebuildAllWrapperCmd")

    ' Create a button control
    Call oPanel.CommandControls.AddButton(oDef3, True)
End Sub
```

Copy Code

```
    ' Get the ribbon associated with the part document
    Dim oPartRibbon As Ribbon
    oPartRibbon = ThisApplication.UserInterfaceManager.Ribbons.Item("Part")

    ' Get the "Tools" tab
    Dim oTab As RibbonTab
    oTab = oPartRibbon.RibbonTabs.Item("id_TabTools")

    ' Create a panel named "Update", positioned after the "Measure" panel in the Tools tab.
    Dim oPanel As RibbonPanel
    oPanel = oTab.RibbonPanels.Add("Update", "ToolsTabUpdatePanel", "SampleClientId", "id_PanelP_ToolsMeasure")

    ' Get the update commands
    Dim oDef1 As ButtonDefinition
    oDef1 = ThisApplication.CommandManager.ControlDefinitions.Item("AppLocalUpdateCmd")

    Dim oDef2 As ButtonDefinition
    oDef2 = ThisApplication.CommandManager.ControlDefinitions.Item("AppUpdateMassPropertiesCmd")

    Dim oDefs As ObjectCollection
    oDefs = ThisApplication.TransientObjects.CreateObjectCollection

    oDefs.Add(oDef1)
    oDefs.Add(oDef2)

    ' Create a split button control
    Call oPanel.CommandControls.AddSplitButton(oDef1, oDefs, True)

    ' Get the rebuild command
    Dim oDef3 As ButtonDefinition
    oDef3 = ThisApplication.CommandManager.ControlDefinitions.Item("AppRebuildAllWrapperCmd")

    ' Create a button control
    Call oPanel.CommandControls.AddButton(oDef3, True)
```

Window Selection

Description

This sample demonstrates using the selection events to window-select multiple edges. Selection is dependent on events and VB only supports events within a class module.

Code Samples

- VBA
- iLogic

To use the sample copy the TestWindowSelection sub into a code module. Create a new class module called clsSelect and copy all of the rest of the code into it. To run the sample, have a part document open that contains some geometry and run the TestWindowSelection sub.

Copy Code

```
Private oSelect As clsSelect

Public Sub TestWindowSelection()
    ' Create a new clsSelect object.
    Set oSelect = New clsSelect

    ' Call the WindowSelect method of the clsSelect object
    oSelect.WindowSelect
End Sub

'************************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsSelect". The name can be
' changed but you'll need to change the declaration in the
' calling function "TestWindowSelection" to use the new name.

' Declare the event objects
Private WithEvents oInteractEvents As InteractionEvents
Private WithEvents oSelectEvents As SelectEvents

' Declare a flag that's used to determine if command prompts are shown as tooltips.
Private bTooltipEnabled As Boolean

Public Function WindowSelect()
    ' Create an InteractionEvents object.
    Set oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

    ' Ensure interaction is enabled.
    oInteractEvents.InteractionDisabled = False

    ' Set a reference to the select events.
    Set oSelectEvents = oInteractEvents.SelectEvents
```

```
    ' Set the filter for circular edges (this includes circular arcs).
    oSelectEvents.AddSelectionFilter kPartEdgeCircularFilter

    oSelectEvents.WindowSelectEnabled = True

    bTooltipEnabled = ThisApplication.GeneralOptions.ShowCommandPromptTooltips
    ThisApplication.GeneralOptions.ShowCommandPromptTooltips = True

    oInteractEvents.StatusBarText = "Window select. Esc to exit."

    ' Start the InteractionEvents object.
    oInteractEvents.Start
End Function

Private Sub oInteractEvents_OnTerminate()
    ' Reset to original value
    ThisApplication.GeneralOptions.ShowCommandPromptTooltips = bTooltipEnabled

    ' Clean up.
    Set oSelectEvents = Nothing
    Set oInteractEvents = Nothing
End Sub

Private Sub oSelectEvents_OnPreSelect(PreSelectEntity As Object, DoHighlight As Boolean, MorePreSelectEntities As ObjectCollection, By'
    ' Set a reference to the selected edge.
    ' Only circular edges can come through since the circular edge filter was set.
    Dim oEdge As Edge
    Set oEdge = PreSelectEntity

    ' Allow only fully circular edges to be picked.
    If Not oEdge.GeometryType = kCircleCurve Then
      DoHighlight = False
    End If
End Sub

Private Sub oSelectEvents_OnSelect(ByVal JustSelectedEntities As ObjectsEnumerator, ByVal SelectionDevice As SelectionDeviceEnum, ByVa
    MsgBox "Picked " & JustSelectedEntities.Count & " circular edges."
End Sub
```

To run the sample, have a part document open that contains some geometry and run the sample.

Copy Code

```
Class Test
        Private oSelect As clsSelect

        Sub Main
            ' Create a new clsSelect object.
            oSelect = New clsSelect
                oSelect.ThisApplication = ThisApplication
            ' Call the WindowSelect method of the clsSelect object
            oSelect.WindowSelect
        End Sub
End Class

Class clsSelect
        ' Declare the event objects
        Private WithEvents oInteractEvents As InteractionEvents
        Private WithEvents oSelectEvents As SelectEvents
        Public ThisApplication As Inventor.Application

        ' Declare a flag that's used to determine if command prompts are shown as tooltips.
        Private bTooltipEnabled As Boolean

        Public Function WindowSelect()
            ' Create an InteractionEvents object.
            oInteractEvents = ThisApplication.CommandManager.CreateInteractionEvents

            ' Ensure interaction is enabled.
            oInteractEvents.InteractionDisabled = False

            ' Set a reference to the select events.
            oSelectEvents = oInteractEvents.SelectEvents

            ' Set the filter for circular edges (this includes circular arcs).
            oSelectEvents.AddSelectionFilter( kPartEdgeCircularFilter)

            oSelectEvents.WindowSelectEnabled = True

            bTooltipEnabled = ThisApplication.GeneralOptions.ShowCommandPromptTooltips
            ThisApplication.GeneralOptions.ShowCommandPromptTooltips = True

            oInteractEvents.StatusBarText = "Window select. Esc to exit."

            ' Start the InteractionEvents object.
            oInteractEvents.Start
        End Function

        Private Sub oInteractEvents_OnTerminate() Handles oInteractEvents.OnTerminate
            ' Reset to original value
            ThisApplication.GeneralOptions.ShowCommandPromptTooltips = bTooltipEnabled

            ' Clean up.
            oSelectEvents = Nothing
            oInteractEvents = Nothing
        End Sub

        Private Sub oSelectEvents_OnPreSelectOnPreSelect(ByRef PreSelectEntity As Object, ByRef DoHighlight As Boolean, ByRef MorePreS
            ' Set a reference to the selected edge.
            ' Only circular edges can come through since the circular edge filter was set.
            Dim oEdge As Edge
            oEdge = PreSelectEntity

            ' Allow only fully circular edges to be picked.
            If Not oEdge.GeometryType = kCircleCurve Then
              DoHighlight = False
            End If
```

```
        End Sub

        Private Sub oSelectEvents_OnSelect(ByVal JustSelectedEntities As ObjectsEnumerator, ByVal SelectionDevice As SelectionDeviceEn
            MsgBox ("Picked " & JustSelectedEntities.Count & " circular edges.")
        End Sub

End Class
```

Cancel a double click

Description

Demonstrates how to receive (and in this case, cancel) a double click from a user. Run the sample code and double click a browser node to trigger the event.

Code Samples

- VBA
- iLogic

Copy Code

```
Option Explicit
   Private WithEvents oUIEvents As UserInputEvents

   Private Sub Class_Initialize()
       Set oUIEvents = ThisApplication.CommandManager.UserInputEvents
   End Sub

Private Sub oUIEvents_OnDoubleClick(ByVal SelectedEntities As ObjectsEnumerator, ByVal SelectionDevice As SelectionDeviceEnum, ByVal B
       Debug.Print "OnDoubleClick: "; SelectedEntities.Count; SelectionDevice; ShiftKeys; View.Caption
       If SelectionDevice = kBrowserSelection Then
           MsgBox "Hello there! I'm not letting you activate this"
           ThisApplication.CommandManager.ControlDefinitions("AppAboutInventorCmd").Execute2 (False)
           HandlingCode = kEventCanceled
       End If
   End Sub
```

Copy Code

```
Class Test
        Private WithEvents oUIEvents As UserInputEvents

        Sub Main
                oUIEvents = ThisApplication.CommandManager.UserInputEvents
        End Sub

        Private Sub oUIEvents_OnDoubleClick(SelectedEntities As Inventor.ObjectsEnumerator, SelectionDevice As Inventor.SelectionDevic
               Logger.Info("OnDoubleClick: "& SelectedEntities.Count & SelectionDevice & ShiftKeys & View.Caption)

               If SelectionDevice = kBrowserSelection Then
                   MsgBox( "Hello there! I'm not letting you activate this")
                   ThisApplication.CommandManager.ControlDefinitions("AppAboutInventorCmd").Execute2 (False)
                   HandlingCode = kEventCanceled
               End If
        End Sub
End Class
```

OnDrag Event - dragging a WorkPoint

Description

This sample demonstrates the use of the OnDrag event to drag fixed work points when no command is active. This sample only allows drags parallel to the X-Y plane. This sample is dependent on events and VB only supports events within a class module.

Code Samples

- VBA
- iLogic

To use the sample copy the WorkPointDrag sub into a code module. Create a new class module called clsDragWorkPoint and copy all of the rest of the code into it. Have a part document open that contains at least one fixed work point. Run the sample and drag the fixed work point.

Copy Code

```
Option Explicit
Public oDragWorkPoint As clsDragWorkPoint

Sub WorkPointDrag()

    Set oDragWorkPoint = New clsDragWorkPoint
    oDragWorkPoint.Initialize

End Sub

'************************************************************
' The declarations and functions below need to be copied into
' a class module whose name is "clsDragWorkPoint".  The name
' can be changed but you'll need to change the declaration in
```

```
' the calling function "WorkPointDrag" to use the new name.
Option Explicit
Private WithEvents oUserInputEvents As UserInputEvents
Private oIE As InteractionEvents
Private WithEvents oMouseEvents As MouseEvents
Private oIntGraphics As InteractionGraphics
Private oWP As WorkPoint

Public Sub Initialize()
    Set oUserInputEvents = ThisApplication.CommandManager.UserInputEvents
End Sub

Private Sub oUserInputEvents_OnDrag(ByVal DragState As Inventor.DragStateEnum, ByVal ShiftKeys As Inventor.ShiftStateEnum, ByVal Model

    Dim oSS As SelectSet
    Set oSS = ThisApplication.ActiveDocument.SelectSet

    If DragState = kDragStateDragHandlerSelection Then

        If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then

            Set oWP = oSS.Item(1)

            If oWP.DefinitionType = kFixedWorkPoint Then
                HandlingCode = kEventCanceled
                Set oIE = ThisApplication.CommandManager.CreateInteractionEvents
                Set oMouseEvents = oIE.MouseEvents
                oMouseEvents.MouseMoveEnabled = True

                Set oIntGraphics = oIE.InteractionGraphics
                Call oIE.SetCursor(kCursorBuiltInCommonSketchDrag)
                oIE.Start
            End If
        End If
    End If
End Sub

Private Sub oMouseEvents_OnMouseMove(ByVal Button As MouseButtonEnum, ByVal ShiftKeys As ShiftStateEnum, ByVal ModelPosition As Point,
    Dim oSS As SelectSet
    Set oSS = ThisApplication.ActiveDocument.SelectSet

    If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then

        Dim oWPDef As FixedWorkPointDef
        Set oWPDef = oWP.Definition

        Dim oProjectedPoint As Inventor.Point
        Call ProjectPoint(ModelPosition, oWPDef.Point, oProjectedPoint)

        ' Set a reference to the transient geometry object for user later.
        Dim oTransGeom As TransientGeometry
        Set oTransGeom = ThisApplication.TransientGeometry

        ' Create a graphics data set object.  This object contains all of the
        ' information used to define the graphics.
        Dim oDataSets As GraphicsDataSets
        Set oDataSets = oIntGraphics.GraphicsDataSets

        If oDataSets.Count <> 0 Then
            oDataSets.Item(1).Delete
        End If

        ' Create a coordinate set.
        Dim oCoordSet As GraphicsCoordinateSet
        Set oCoordSet = oDataSets.CreateCoordinateSet(1)

        ' Create an array that contains coordinates that define a set
        ' of outwardly spiraling points.
        Dim oPointCoords(1 To 3) As Double
        ' Define the X, Y, and Z components of the point.
        oPointCoords(1) = oProjectedPoint.X
        oPointCoords(2) = oProjectedPoint.Y
        oPointCoords(3) = oProjectedPoint.Z

        ' Assign the points into the coordinate set.
        Call oCoordSet.PutCoordinates(oPointCoords)

        ' Create the ClientGraphics object.
        Dim oClientGraphics As ClientGraphics
        Set oClientGraphics = oIntGraphics.PreviewClientGraphics

        If oClientGraphics.Count <> 0 Then
            oClientGraphics.Item(1).Delete
        End If

        ' Create a new graphics node within the client graphics objects.
        Dim oPtNode As GraphicsNode
        Set oPtNode = oClientGraphics.AddNode(1)

        ' Create a PointGraphics object within the node.
        Dim oPtGraphics As PointGraphics
        Set oPtGraphics = oPtNode.AddPointGraphics

        ' Assign the coordinate set to the line graphics.
        oPtGraphics.CoordinateSet = oCoordSet
        oPtGraphics.PointRenderStyle = kCrossPointStyle
        ThisApplication.ActiveView.Update
    End If
End Sub

Private Sub oMouseEvents_OnMouseUp(ByVal Button As MouseButtonEnum, ByVal ShiftKeys As ShiftStateEnum, ByVal ModelPosition As Point, B

    Dim oSS As SelectSet
    Set oSS = ThisApplication.ActiveDocument.SelectSet

    If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then
```

```
            Dim oWPDef As FixedWorkPointDef
            Set oWPDef = oWP.Definition

            Dim oProjectedPoint As Inventor.Point
            Call ProjectPoint(ModelPosition, oWPDef.Point, oProjectedPoint)

            ' Reposition the fixed work point
            oWPDef.Point = oProjectedPoint
            ThisApplication.ActiveDocument.Update
            oIE.Stop

            Set oWP = Nothing

    End If
End Sub

' Project the ModelPosition to a plane parallel to the
' X-Y plane on which the work point currently is.
Private Sub ProjectPoint(ByVal ModelPosition As Inventor.Point, ByVal WorkPointPosition As Inventor.Point, ProjectedPoint As Inventor.

    ' Set a reference to the camera object
    Dim oCamera As Inventor.Camera
    Set oCamera = ThisApplication.ActiveView.Camera

    Dim oVec As Vector
    Set oVec = oCamera.Eye.VectorTo(oCamera.Target)

    Dim oLine As Line
    Set oLine = ThisApplication.TransientGeometry.CreateLine(ModelPosition, oVec)

    ' Create the z-axis vector
    Dim oZAxis As Vector
    Set oZAxis = ThisApplication.TransientGeometry.CreateVector(0, 0, 1)

    ' Create a plane parallel to the X-Y plane
    Dim oWPPlane As Plane
    Set oWPPlane = ThisApplication.TransientGeometry.CreatePlane(WorkPointPosition, oZAxis)

    Set ProjectedPoint = oWPPlane.IntersectWithLine(oLine)
End Sub
```

Have a part document open that contains at least one fixed work point. Run the sample and drag the fixed work point.

Copy Code

```
Sub Main
    Dim oDragWorkPoint As clsDragWorkPoint
    oDragWorkPoint = New clsDragWorkPoint
    oDragWorkPoint.Initialize(ThisApplication)

End Sub

Class clsDragWorkPoint
        Private ThisApplication As Inventor.Application
        Private WithEvents oUserInputEvents As UserInputEvents
        Private oIE As InteractionEvents
        Private WithEvents oMouseEvents As MouseEvents
        Private oIntGraphics As InteractionGraphics
        Private oWP As WorkPoint
        Private bStop As Boolean

        Public Sub Initialize(oApp As Inventor.Application)
                ThisApplication = oApp
            oUserInputEvents = ThisApplication.CommandManager.UserInputEvents

                ' Loop until a selection is made.
'           Do
'               ThisApplication.UserInterfaceManager.DoEvents
'           Loop Until bStop
        End Sub

        Private Sub oUserInputEvents_OnDrag(ByVal DragState As Inventor.DragStateEnum, ByVal ShiftKeys As Inventor.ShiftStateEnum, ByV

            Dim oSS As SelectSet
            oSS = ThisApplication.ActiveDocument.SelectSet

            If DragState = kDragStateDragHandlerSelection Then

                If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then

                    oWP = oSS.Item(1)

                    If oWP.DefinitionType = kFixedWorkPoint Then

                        HandlingCode = kEventCanceled
                        oIE = ThisApplication.CommandManager.CreateInteractionEvents
                        oMouseEvents = oIE.MouseEvents
                        oMouseEvents.MouseMoveEnabled = True

                        oIntGraphics = oIE.InteractionGraphics
                        Call oIE.SetCursor(kCursorBuiltInCommonSketchDrag)
                        oIE.Start
                    End If
                End If
            End If
        End Sub

        Private Sub oMouseEvents_OnMouseMove(ByVal Button As MouseButtonEnum, ByVal ShiftKeys As ShiftStateEnum, ByVal ModelPosition A
            Dim oSS As SelectSet
            oSS = ThisApplication.ActiveDocument.SelectSet

            If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then

                Dim oWPDef As FixedWorkPointDef
                oWPDef = oWP.Definition

                Dim oProjectedPoint As Inventor.Point
                Call ProjectPoint(ModelPosition, oWPDef.Point, oProjectedPoint)
```

```
            ' Set a reference to the transient geometry object for user later.
            Dim oTransGeom As TransientGeometry
            oTransGeom = ThisApplication.TransientGeometry

            ' Create a graphics data set object.  This object contains all of the
            ' information used to define the graphics.
            Dim oDataSets As GraphicsDataSets
            oDataSets = oIntGraphics.GraphicsDataSets

            If oDataSets.Count <> 0 Then
                oDataSets.Item(1).Delete
            End If

            ' Create a coordinate set.
            Dim oCoordSet As GraphicsCoordinateSet
            oCoordSet = oDataSets.CreateCoordinateSet(1)

            ' Create an array that contains coordinates that define a set
            ' of outwardly spiraling points.
            Dim oPointCoords(0 To 2) As Double
            ' Define the X, Y, and Z components of the point.
            oPointCoords(0) = oProjectedPoint.X
            oPointCoords(1) = oProjectedPoint.Y
            oPointCoords(2) = oProjectedPoint.Z

            ' Assign the points into the coordinate set.
            Call oCoordSet.PutCoordinates(oPointCoords)

            ' Create the ClientGraphics object.
            Dim oClientGraphics As ClientGraphics
            oClientGraphics = oIntGraphics.PreviewClientGraphics

            If oClientGraphics.Count <> 0 Then
                oClientGraphics.Item(1).Delete
            End If

            ' Create a new graphics node within the client graphics objects.
            Dim oPtNode As GraphicsNode
            oPtNode = oClientGraphics.AddNode(oClientGraphics.Count + 1) 'oClientGraphics.AddNode(1)

            ' Create a PointGraphics object within the node.
            Dim oPtGraphics As PointGraphics
            oPtGraphics = oPtNode.AddPointGraphics

            ' Assign the coordinate set to the line graphics.
            oPtGraphics.CoordinateSet = oCoordSet
            oPtGraphics.PointRenderStyle = kCrossPointStyle
            ThisApplication.ActiveView.Update
        End If
    End Sub

    Private Sub oMouseEvents_OnMouseUp(ByVal Button As MouseButtonEnum, ByVal ShiftKeys As ShiftStateEnum, ByVal ModelPosition As

        Dim oSS As SelectSet
        oSS = ThisApplication.ActiveDocument.SelectSet

        If oSS.Count = 1 And oSS.Item(1).Type = kWorkPointObject Then

                Dim oWPDef As FixedWorkPointDef
                oWPDef = oWP.Definition

                Dim oProjectedPoint As Inventor.Point
                Call ProjectPoint(ModelPosition, oWPDef.Point, oProjectedPoint)

                ' Reposition the fixed work point
                oWPDef.Point = oProjectedPoint
                ThisApplication.ActiveDocument.Update
                oIE.Stop

                oWP = Nothing

        End If
    End Sub

    ' Project the ModelPosition to a plane parallel to the
    ' X-Y plane on which the work point currently is.
    Private Sub ProjectPoint(ByVal ModelPosition As Inventor.Point, ByVal WorkPointPosition As Inventor.Point, ByRef ProjectedPoin

        ' Set a reference to the camera object
        Dim oCamera As Inventor.Camera
        oCamera = ThisApplication.ActiveView.Camera

        Dim oVec As Vector
        oVec = oCamera.Eye.VectorTo(oCamera.Target)

        Dim oLine As Line
        oLine = ThisApplication.TransientGeometry.CreateLine(ModelPosition, oVec)

        ' Create the z-axis vector
        Dim oZAxis As Vector
        oZAxis = ThisApplication.TransientGeometry.CreateVector(0, 0, 1)

        ' Create a plane parallel to the X-Y plane
        Dim oWPPlane As Plane
        oWPPlane = ThisApplication.TransientGeometry.CreatePlane(WorkPointPosition, oZAxis)

        ProjectedPoint = oWPPlane.IntersectWithLine(oLine)
    End Sub
End Class
```

Move view tab between different view frames

Description

This sample demonstrates how to move views using ViewTab between different view frames.

Code Samples

- VBA
- iLogic

This sample demonstrates how to move a View via its ViewTab to generate a new ViewFrame and move it to another ViewFrame.

Copy Code

```
Sub MoveViewTabSample()
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oView1 As View
    Set oView1 = oDoc.Views(1)

    Dim oViewTab1 As ViewTab
    Set oViewTab1 = oView1.ViewTab


    Dim oView2 As View
    Set oView2 = oDoc.Views.Add

    Dim oViewTab2 As ViewTab
    Set oViewTab2 = oView2.ViewTab

    ' Move the second View to a new ViewFrame
    Dim oViewFrame As ViewFrame
    Set oViewFrame = oViewTab2.MoveToNewViewFrame(500, 600, 200, 100)

    Dim oDoc1 As PartDocument
    Set oDoc1 = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oView3 As View
    Set oView3 = oDoc1.Views(1)

    Dim oViewTab3 As ViewTab
    Set oViewTab3 = oView3.ViewTab

    ' Move the third View back to main frame.
    Call oViewTab3.MoveToGroup(True, oViewTab1, kDockBottom)
End Sub
```

This sample demonstrates how to move a View via its ViewTab to generate a new ViewFrame and move it to another ViewFrame.

Copy Code

```
    Dim oDoc As PartDocument
    oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oView1 As View
    oView1 = oDoc.Views(1)

    Dim oViewTab1 As ViewTab
    oViewTab1 = oView1.ViewTab


    Dim oView2 As View
    oView2 = oDoc.Views.Add

    Dim oViewTab2 As ViewTab
    oViewTab2 = oView2.ViewTab

    ' Move the second View to a new ViewFrame
    Dim oViewFrame As ViewFrame
    oViewFrame = oViewTab2.MoveToNewViewFrame(500, 600, 200, 100)

    Dim oDoc1 As PartDocument
    oDoc1 = ThisApplication.Documents.Add(kPartDocumentObject)

    Dim oView3 As View
    oView3 = oDoc1.Views(1)

    Dim oViewTab3 As ViewTab
    oViewTab3 = oView3.ViewTab

    ' Move the third View back to main frame.
    Call oViewTab3.MoveToGroup(True, oViewTab1, kDockBottom)
```