

# Inventor 2022 API 基礎トレーニング



# Course Agenda

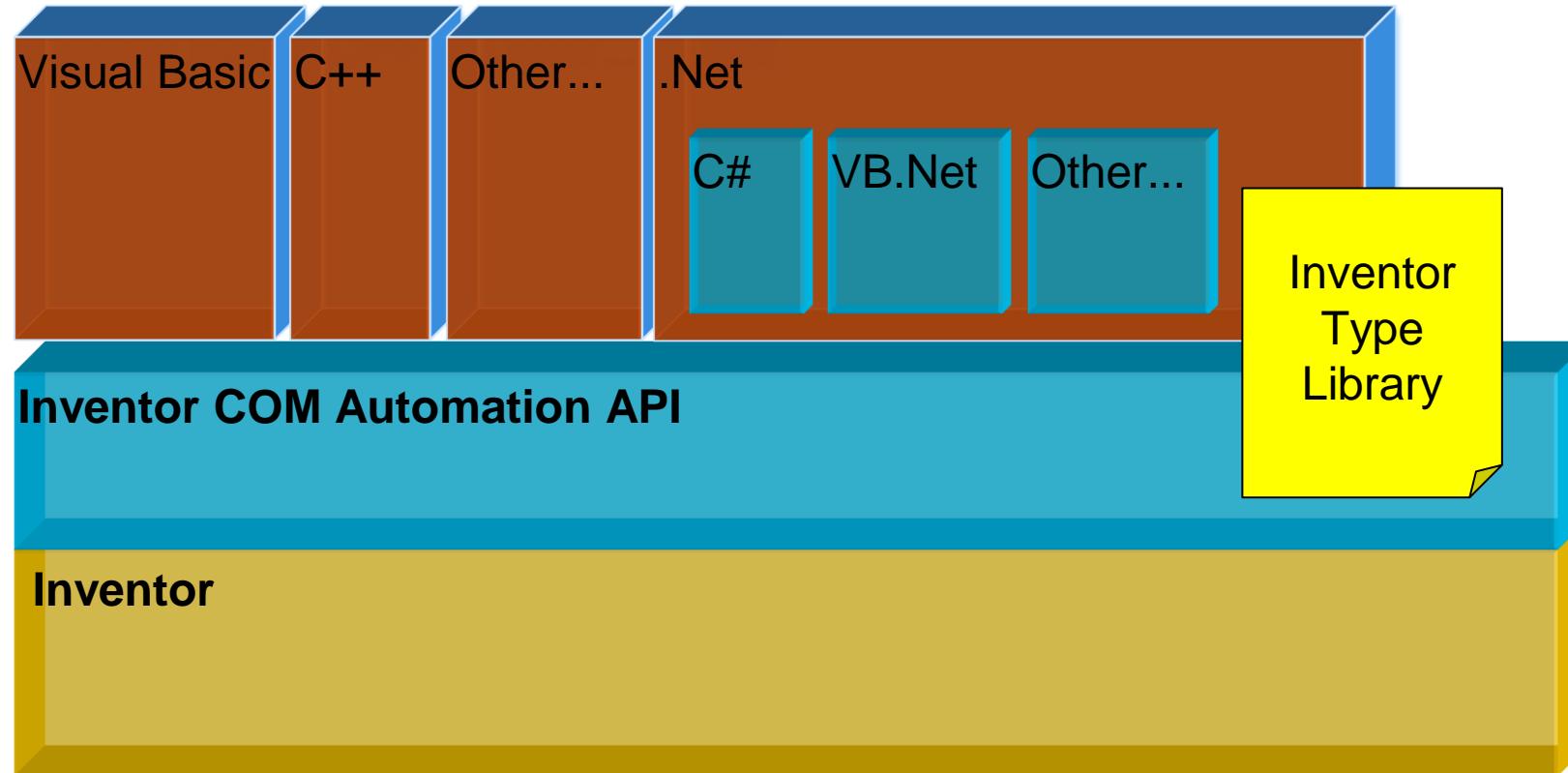
- Inventor APIの概要
- Inventor Object Model
- Application オブジェクト
- Document オブジェクトの概要
- パーツドキュメント
- B-Rep
- シートメタル
- アセンブリ
- Drawingドキュメント
- イベント
- ユーザインターフェース
- 印刷とファイルトランスレーター
- アペレンテス
- クライアントグラフィクス
- 参照キー
- トランザクション
- 演習の解答

# Inventor APIの概要

# Inventor APIの概要

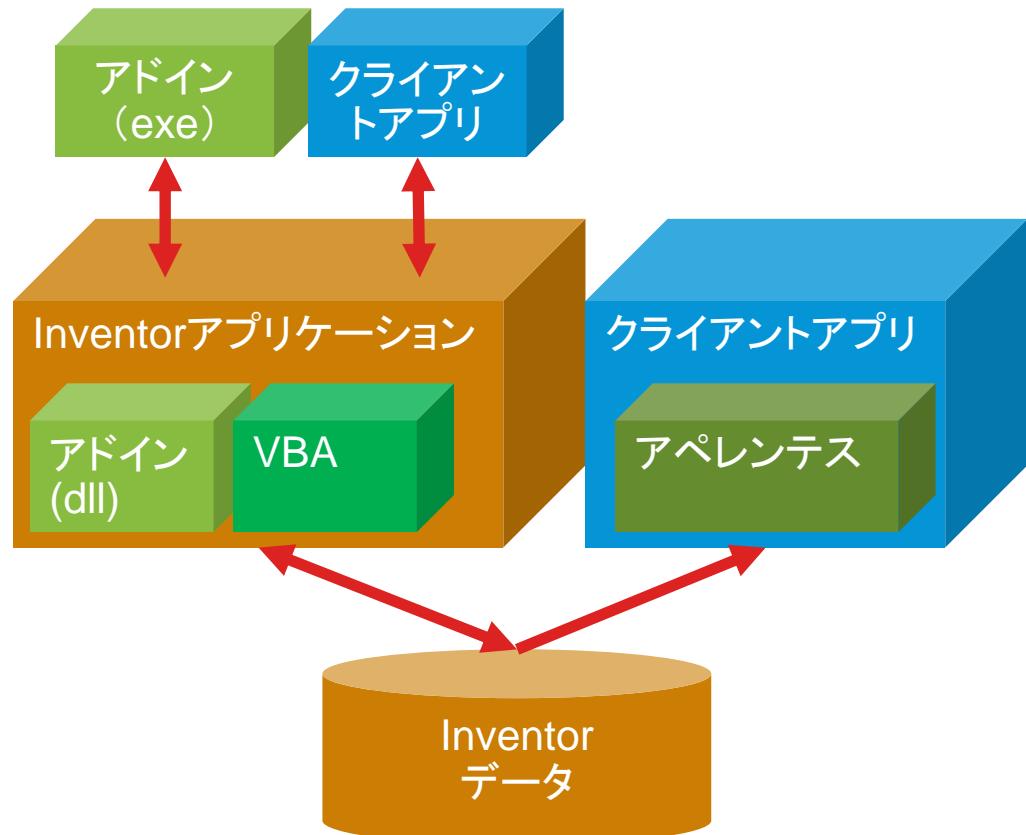
## COM API Model

InventorのAPIは、COM(Component Object Model) Automation interfaceで公開される。



# Inventor APIの概要

## APIアクセス方法



- VBA
- アドインdllまたはexe
- クライアントアプリ
- アペレンテス

# Inventor APIの概要

## VBA(Visual Basic for Applications)

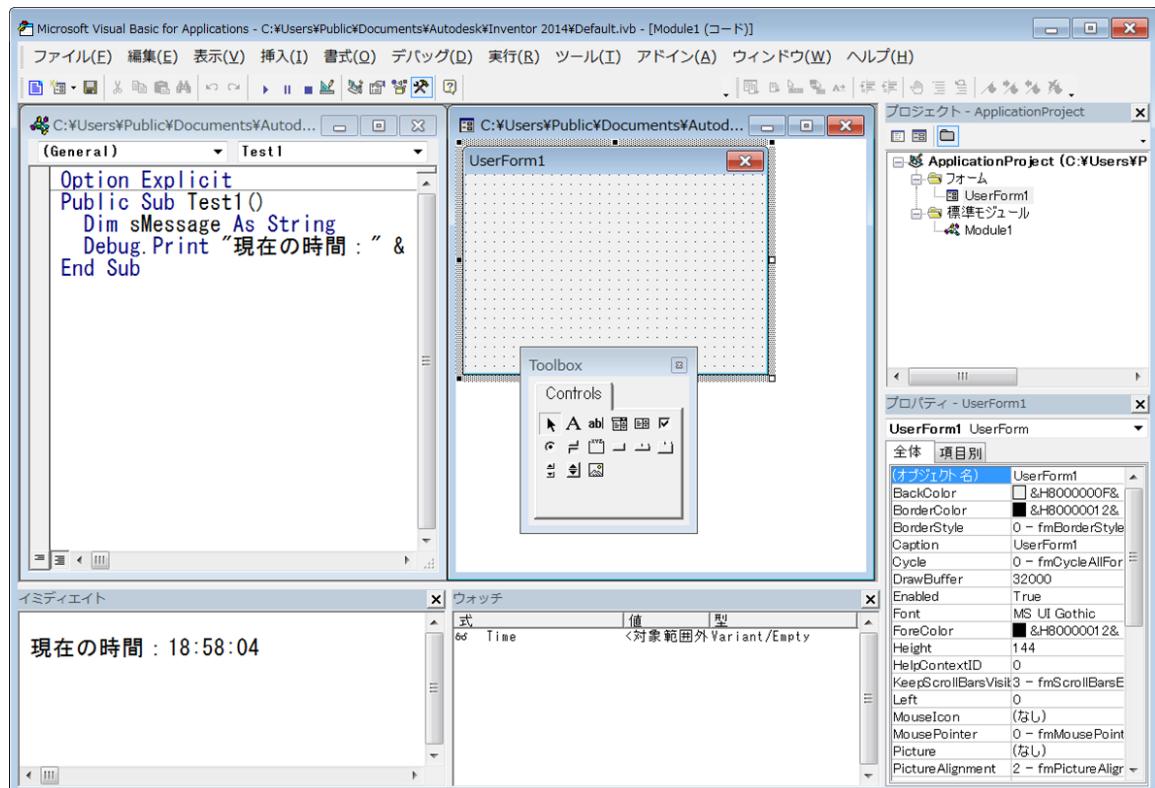
- Visual Basic for Applications (VBA)は、Microsoft により開発されたプログラミング環境
- Inventorの一部として、アプリケーション内に埋め込み無償で供給され、Inventorのデータ内にプログラムを埋め込むことが可能
- Inventorのリボンメニューのツール(T) → Visual Basic Editor(V)、またはAlt-F11で起動
- 主な使用用途は、エンドユーザーが“マクロ”を作るため。作成したマクロは、“マクロ”コマンドまたは、リボン タブの カスタマイズ ダイアログの マクロに登録して実行。(詳細は、製品 オンラインヘルプの リボンタブの カスタマイズ ダイアログのリファレンス”を参照)
- 64BitOS上でネイティブに実行されるため、Microsoftがリリースしているがリリースしている32Bit用のコントロールなどは動作しないので、フォームなどへの組み込みには注意が必要
- インタプリタ言語(コンパイルが不要)なため、実行時にInventorのObject Modelを参照しながら作業が可能

### 注意

旧バージョンのInventorに実装されていた自動マクロ機能(AutoOpen,AutoNew,AutoSave,AutoClose,AutoEdit) はセキュリティー観点より、Inventor2014 製品以降、取り除かれています

# Inventor APIの概要

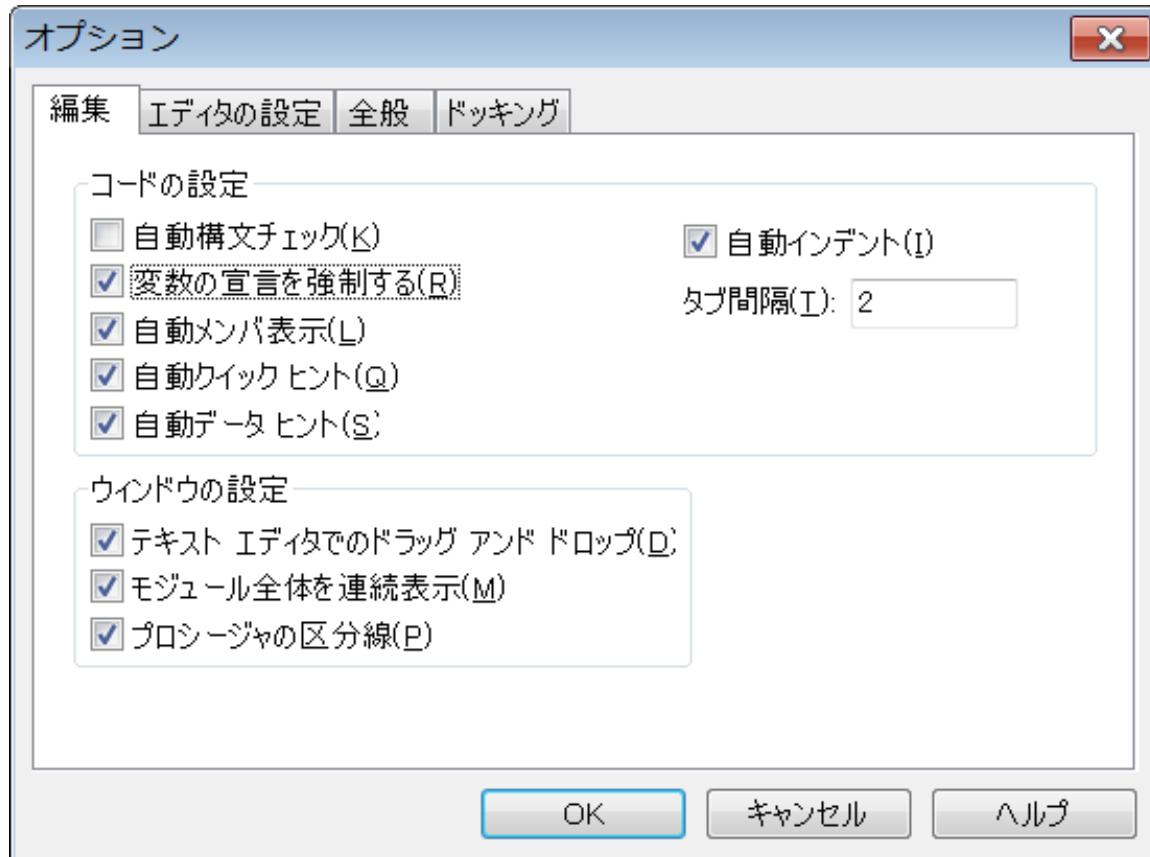
## VBA統合開発環境 VBA IDE (Integrated Development Environment)



- プロジェクトエクスプローラ
- プロパティ
- コードWindow
- フォームWindow
- イミディエイトWindow
- ウオッチWindow

# Inventor APIの概要

## VBAオプション



- 自動構文チェック → Off

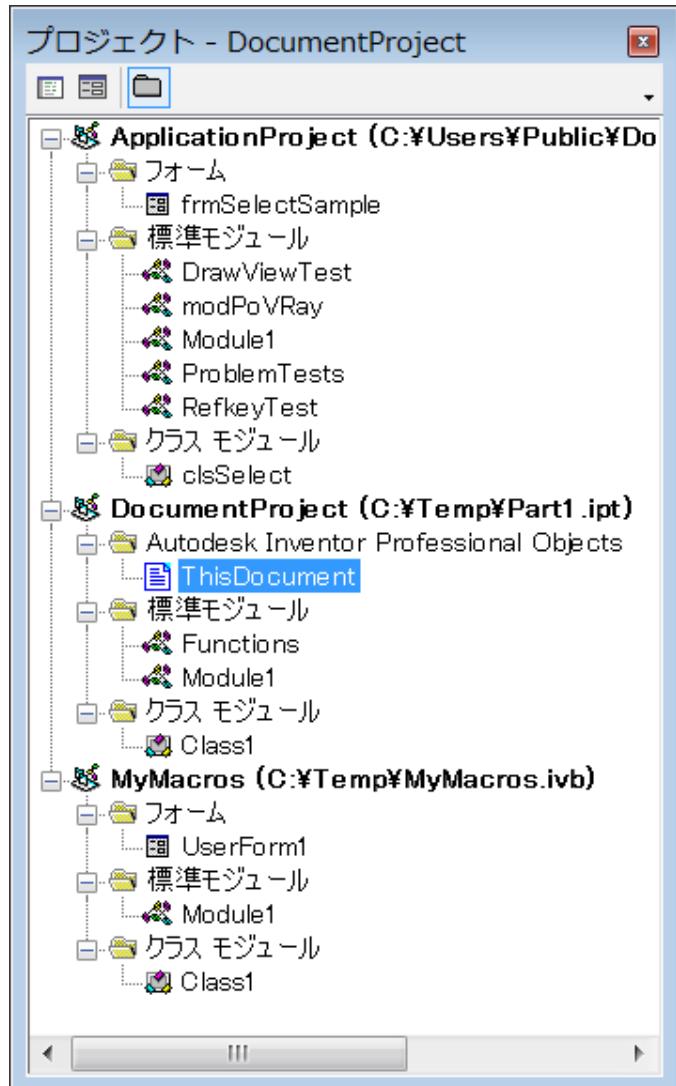
コードをリアルタイムにチェックし、構文エラーがあるとアラートを表示する機能。便利な機能だが、アラートがやや煩雑であることと、構文エラーはソースエディタで赤字で表示されるため、Offにしたほうがコードを記述しやすいため、Offにすることを推奨

- 変数の宣言を強制する → On

VBAでは未定義の変数は、実行時に自動的に変数定義が行われるが、変数名の競合、入力ミス等によるエラーが発生しやすいため、宣言を強制することを推奨

# Inventor APIの概要

## VBAプロジェクト



- プロジェクトとは、プログラムを含むファイルと、モジュールのコレクション(フォーム、コード、クラスモジュール)
- プロジェクトのタイプ(3種類)
  1. Application project … Inventor本体に固定。一つ存在。
  2. Document Project… Inventorの各種ファイルに保存可能。InventorファイルのOpen時に自動でロード。
  3. User Project … 外部.ivbファイルに保存。ロード/アンロードは別途操作が必要。
    - ✓ ファイル→プロジェクトのロード/ロード解除
    - ✓ VBA APIより操作が可能。

# Inventor APIの概要

## Application projectの保管場所

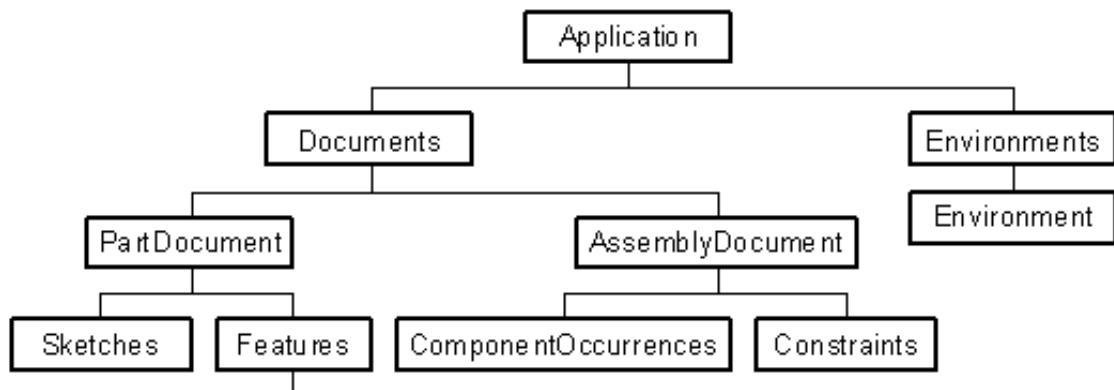


- 外部ファイル(.ivb)として保管
  - ✓ ツール → アプリケーションオプションの「ファイル タブ」既存の VBA プロジェクト

# Inventor APIの概要

## Inventor VBA環境 既定の変数

### □ Inventor API Object Model



- **ThisApplication** 変数(スコープ:グローバル)

- ✓ **Application**オブジェクトの参照を提供

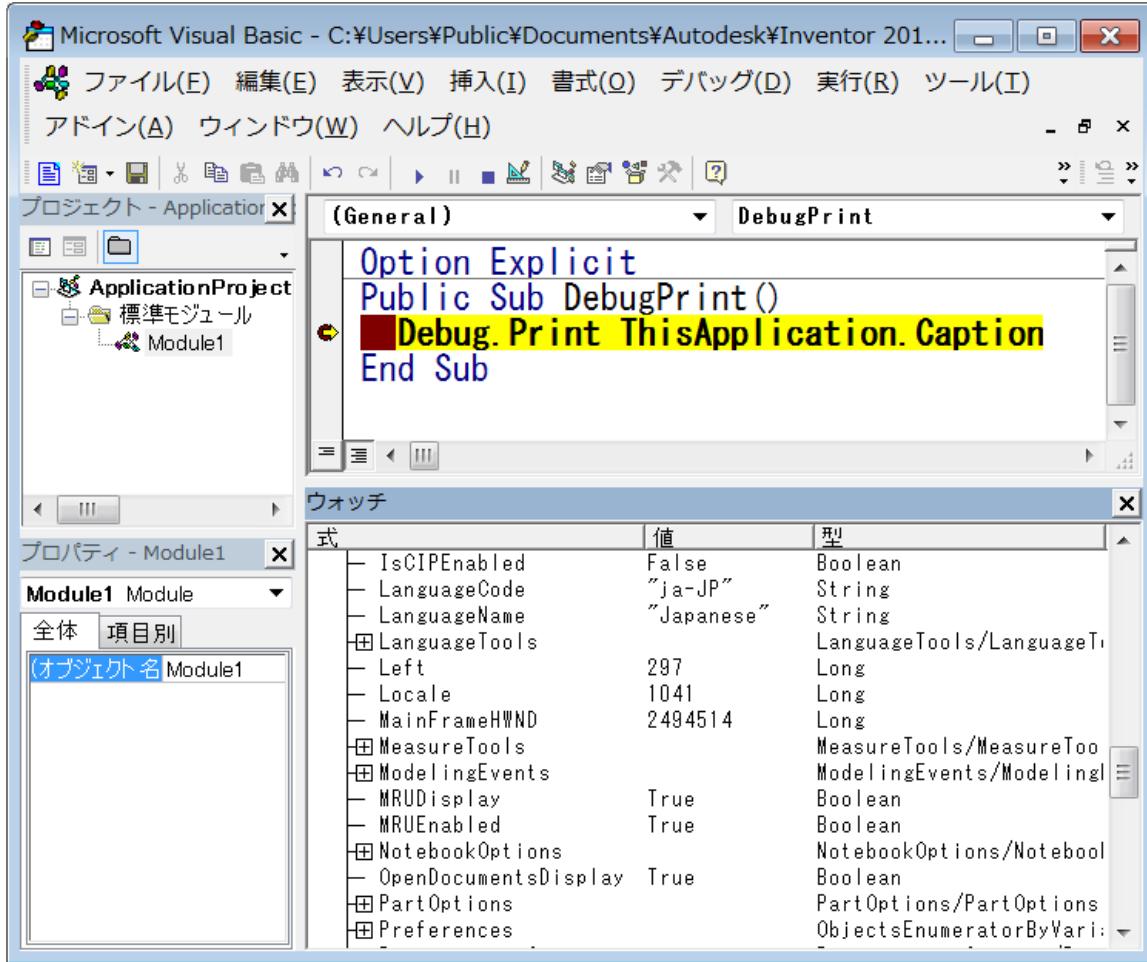
- **Application**オブジェクトはInventor API Object Modelの最上位のオブジェクト

- **ThisDocument** 変数(スコープ:ドキュメントプロジェクト)

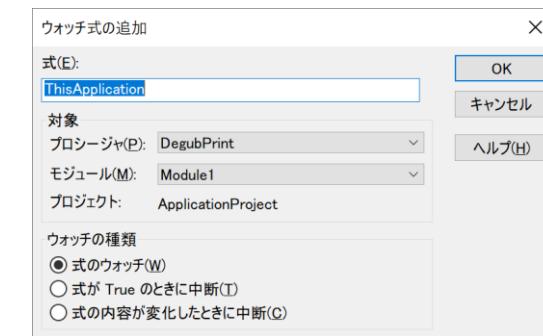
- ✓ プロジェクト内に有るドキュメントの参照を提供

# Inventor APIの概要

## VBA Debuggerの使用



- F9またはコード行の左端の領域をクリックして、行にブレークポイントを追加
- “ThisApplication”変数を右クリックし、ウォッチ式の追加... → “OK” ボタン



- デバッグ停止時に“ウォッチ”ウィンドウで、プロパティの値とオブジェクトの値を展開して参照が可能

# 演習1

## VBA

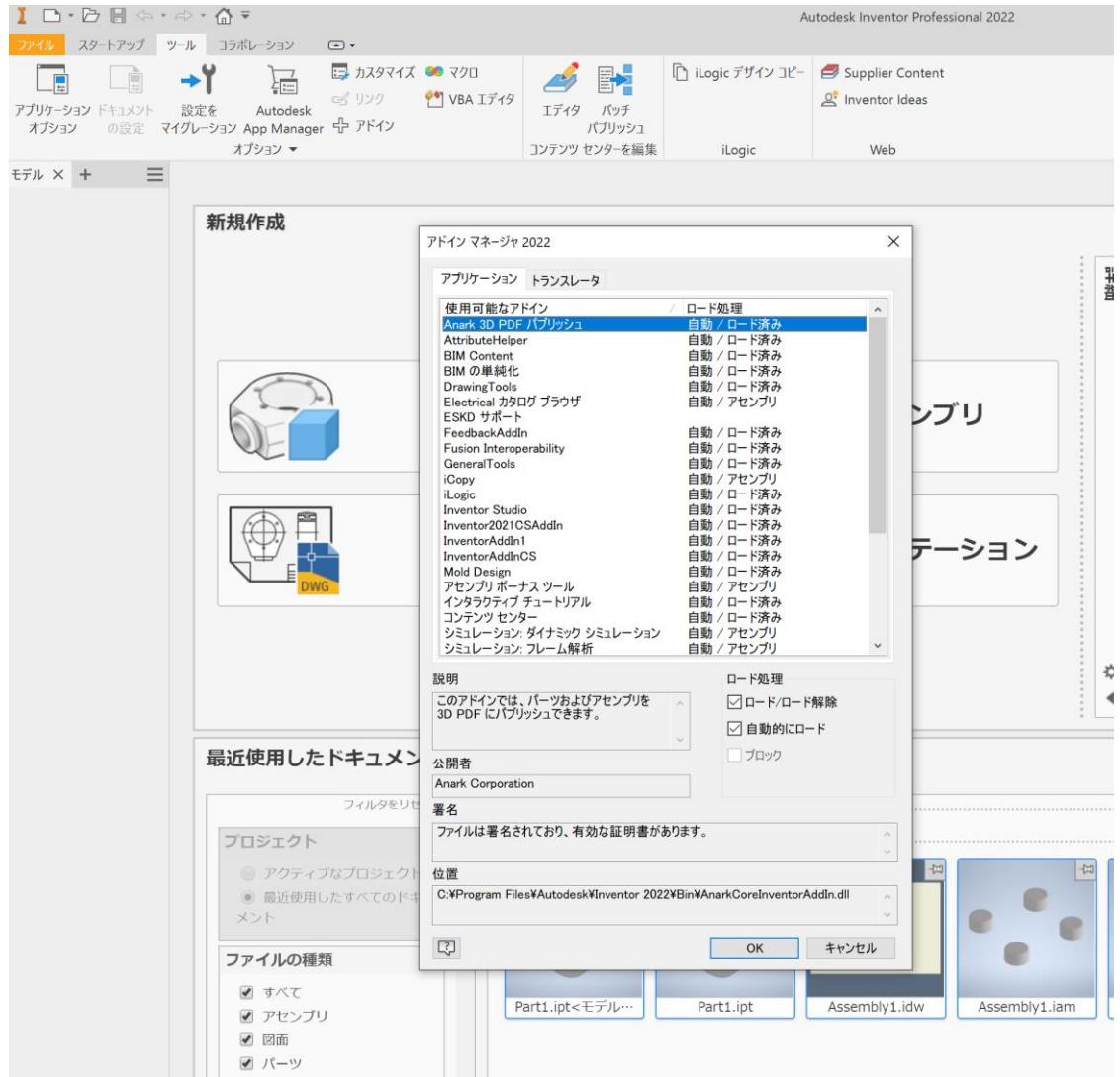
- VBAで、InventorでOpenしてアクティブになっているファイルの、ドキュメントの種別(アセンブリ、パート)およびドキュメント名、ファイル名を表示するコードを作成してください。

- 演習のヒント

- アクティブなドキュメントのオブジェクトは、ApplicationオブジェクトのActiveDocumentプロパティで取得できます
- アクティブなドキュメントのドキュメントの種別は、ApplicationオブジェクトのActiveDocumentTypeプロパティで取得できます
- ドキュメントの種別は、列挙型 DocumentTypeEnum の列挙子で判定できます
- ドキュメントの表示名は、ドキュメントオブジェクトの DisplayName プロパティ、ドキュメントのファイル名は FullFileName プロパティで取得できます

# Inventor APIの概要

## Inventorアドインとは



■Inventorが開始した際に自動的に開始されるCOMコンポーネント。

■ボタン等UIの作成や、Inventorアプリケーションでの様々なイベントに対応した処理を作成可能

# Inventor APIの概要

## InventorアドインとVBAの比較

### ■ Inventorアドインの利点

- ✓ スタートアップ時のロード
- ✓ VB、C#などの新しい開発言語が使用可能(C++、F#、Delphiなども使用可能)
- ✓ InventorのUIに完全に統合
- ✓ 高度なInventorイベントのハンドリング
- ✓ インストーラ開発が可能
- ✓ ソースコード管理が容易
- ✓ ソースコードのセキュリティ
- ✓ より高度なトランザクションのサポート

### ■ VBAの利点

- ✓ 開発ツールが不要(Inventorアプリケーションに包含)
- ✓ 容易なRapidプロトタイプが可能
- ✓ 使いやすいオブジェクトブラウザー
- ✓ Inventor objectを参照しながらのデバッグが容易

# Inventor APIの概要

## レジストリ登録不要(Regfree)なアドイン

- レジストリ登録が不要なCOMコンポーネント
- レジストリに代わり、.addinファイルで、様々なセッティングを定義
- Inventorアプリケーションの所定のディレクトリ内に.addinファイルを置くことでInventorアプリケーション起動時に.addinファイルをロード
- カスタムアドインdllがどこにあるかを.addinファイルに記述。カスタムアドインdllは、任意のフォルダーに置く事が可能

# Inventor APIの概要

## .addinファイルの例

```
<Addin Type="Standard">

<ClassId>{9c93ff52-e2fa-4ec4-8c4a-5747ad0bafef}</ClassId>
<ClientId>{9c93ff52-e2fa-4ec4-8c4a-5747ad0bafef}</ClientId>

<DisplayName>Attribute Helper</DisplayName>
<Description>Utility to manage API attributes.</Description>
<Assembly>AttributeHelper.dll</Assembly>

<LoadOnStartUp>1</LoadOnStartUp>
<UserUnloadable>1</UserUnloadable>
<OSType>Win64</OSType>
<Hidden>0</Hidden>
<SupportedSoftwareVersionGreater Than>
  25..
</SupportedSoftwareVersionGreater Than>
<DataVersion>1</DataVersion>
<UserInterfaceVersion>1</UserInterfaceVersion>

</Addin>
```

- スタートアップ時のロード設定<LoadOnStartUp>  
1:スタートアップ時にロード
- アドイン マネージャでユーザが Add In をロード解除できるかの設定<UserUnloadable>  
1:ロード解除可能
- アドイン マネージャに表示するかを制御す<Hidden>  
0:アドインマネージャに表示
- カスタムアドイン DLL の存在位置<Assembly>  
DLLファイル名を含むパスを指定
- ロードバージョン制御<SupportedSoftwareVersionGreater Than>  
26.. (Inventor 2022以上の場合)
- その他  
SupportedSoftwareVersionLessThan  
SupportedSoftwareVersionGreater Than  
SupportedSoftwareVersionEqualTo
- ロード制御<LoadBehavior>  
0:スタートアップ 1: パーツ 2: アセンブリ 3: プrezentーション  
4: 図面 10: オンデマンド

# Inventor APIの概要

## .dllファイルの配置先と<Assembly>の設定

- 任意のホルダーに、カスタムアドインを配置する場合
  - \*.addin ファイル内の設定は、フルパス付のdll 名  
例)  
c:\temp\AdnAddin.dll
- Inventorの<InstallPath>\binフォルダーに、カスタムアドインを配置する場合
  - \*.addin ファイル内の 設定はdll名のみ  
例)  
AdnAddin.dll
- Inventorの<InstallPath>\bin 直下に作成した任意フォルダー「XXX」に、カスタムアドインを配置する場合
  - \*.addinファイル内の<Assembly>設定は、「任意フォルダ一名」+ “\” + dll名  
例)  
XXX\AdnAddin.dll

# Inventor APIの概要

## .addinファイルの配置先

- バージョンに非依存(.addinファイルの SupportedSoftwareVersionxxx の記述を無視)

C:\ProgramData\Autodesk\Inventor Addins\

- バージョンに依存(.addinファイルの SupportedSoftwareVersionxxx の記述が有効)

C:\ProgramData\Autodesk\Inventor\<Inventor version>\Addins\

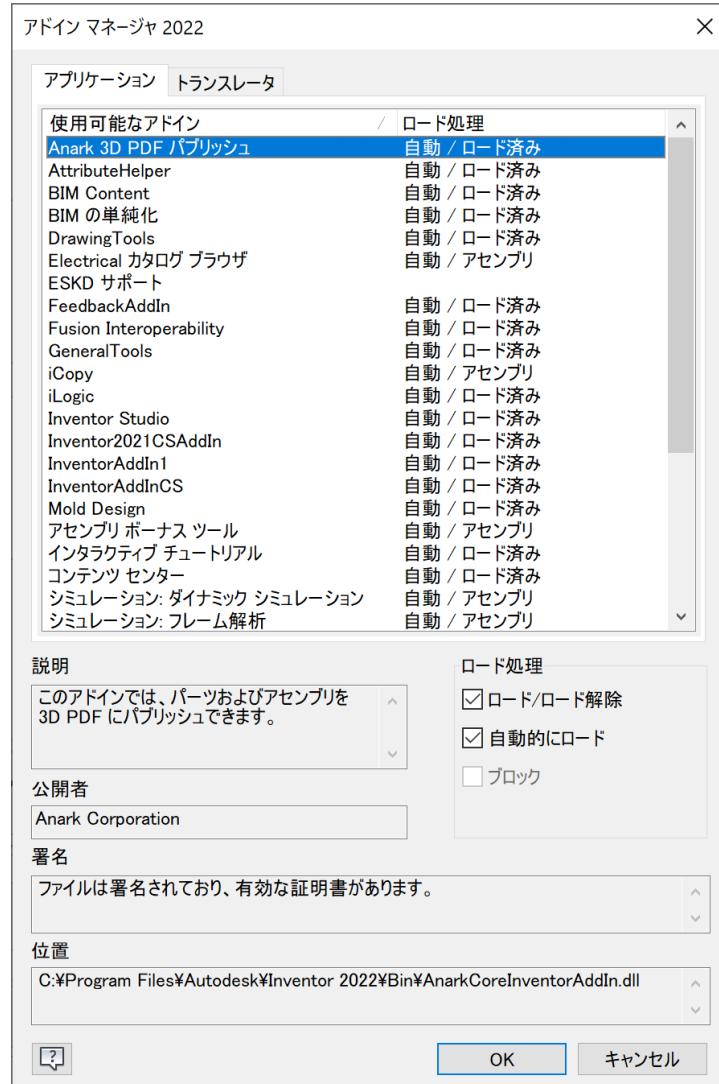
- ユーザー毎にオーバーライド

C:\Users\<user>\<AppData\Roaming\Autodesk\<Inventor version>\Addins\

アドインの登録解除は、.addinファイルを配置したディレクトリより**Autodesk.<AddInName>.Inventor.addin**ファイルを削除

# Inventor APIの概要

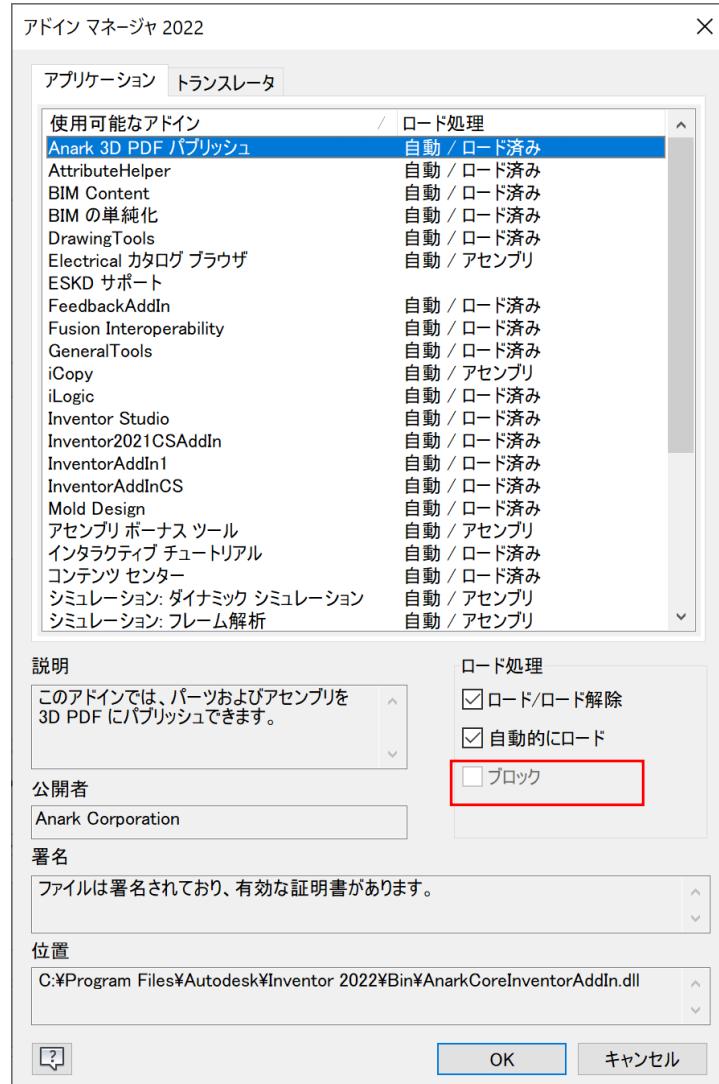
## アドインロードの振る舞い



- .addinファイルの<LoadBehavior>タグで設定
  - 即時(起動時)
  - スタートアップでロード(非推奨)
  
- 目的のタイプのドキュメントが、最初にオープンされる時にロード
  - パーツ
  - アセンブリ
  - プrezentation
  - 図面
- etc.

# Inventor APIの概要

## アドインのデジタル署名



■デジタル署名の無いアドインは、アドインマネージャにより「ブロック」される

■デジタル署名の無いアドインでも、一度「許可」をすれば、次回起動時以降はロックされない

# Inventor APIの概要

## レジストリ登録アドイン

- Inventor 2022から、レジストリ登録を行う形式で作成されたアドインは、Inventorの Add-Ins マネージャ ダイアログに表示されなくなる。
- Inventor 2022でレジストリ登録を行う形式で作成されたアドインを引き続き使用する場合、レジストリキーの値を作成して有効にする必要があります。

キーの場所:

[HKEY\_CURRENT\_USER\SOFTWARE\Autodesk\Inventor\RegistryVersion26.0\System\Preferences]

キー値の名前: LoadRegisterBasedAddins

キー値データ: dword:00000001

- レジストリ登録が必要なアドインは、将来のリリースでサポートされなくなる可能性があるため、Regfree形式への変換を行うことを強く推奨。

# Inventor APIの概要

## Visual Studio アドインWizard



■ Visual Studioで、Inventor Addinを作成するためのコードテンプレート、ビルド設定、Inventor APIライブラリへの参照設定等を含む、Visual Studioのプロジェクトを生成するヘルパーツール

- Addin Wizard(を含むDeveloper Tool)のインストーラーの場所
- C:\Users\Public\Documents\Autodesk\Inventor <version>\SDK
- 配下のDeveloperTools.msi
- インストール手順
  - Visual Studio professional 2015以降がインストールされていることを確認
  - 管理者権限でコマンドプロンプトを起動
  - コマンドプロンプトよりmsiexec.exeを /iオプションで起動しインストーラを起動
  - 例)  
msiexec.exe /i <DeveloperTools.msi ファイルパス>
- アドインの作成
  - Visual Studio を起動し、プロジェクトの新規作成から、Inventor 2022のAddinテンプレートを選択(C++, VB、C#のいずれかが使用可能)

# Inventor APIの概要

## Visual Studio アドインWizardで生成されたテンプレートソースコード

```
<ProgIdAttribute("InventorAddIn2.StandardAddInServer"), _  
GuidAttribute("8b801945-a84a-4e48-a572-3fbff56b9ca9")> _  
1 個の参照  
Public Class StandardAddInServer  
    Implements Inventor.ApplicationAddInServer  
  
    Private WithEvents m_uiEvents As UserInterfaceEvents  
    Private WithEvents m_sampleButton As ButtonDefinition  
  
#Region "ApplicationAddInServer Members"  
  
    ' This method is called by Inventor when it loads the AddIn. The AddIn  
    ' is passed to the Inventor Application object. The FirstTime flag indicates if  
    ' this is the first time. However, with the introduction of the ribbon this ar  
0 個の参照  
Public Sub Activate(ByVal addInSiteObject As Inventor.ApplicationAddIn)  
  
    ' This method is called by Inventor when the AddIn is unloaded. The AddIn  
    ' is unloaded either manually by the user or when the Inventor session is  
0 個の参照  
Public Sub Deactivate() Implements Inventor.ApplicationAddInServer.De  
  
    ' This property is provided to allow the AddIn to expose an API of its  
    ' programs. Typically, this would be done by implementing the AddIn's  
    ' interface in a class and returning that class object through this prop  
0 個の参照  
Public ReadOnly Property Automation() As Object Implements Inventor.A  
  
    ' Note: this method is now obsolete, you should use the  
    ' ControlDefinition functionality for implementing commands.  
0 個の参照  
Public Sub ExecuteCommand(ByVal commandID As Integer) Implements Inventor  
  
#End Region
```

- Activate

addinがInventorにロードされた際に呼び出されるメソッド。主にアドインで実装する処理であるコマンドや、コマンドを起動するためのユーザインターフェイス(リボンのボタン等)の構築・登録の処理を記述

- Deactivate

addinがアンロードされた際に呼び出されるメソッド。主にActivateメソッドで構築・登録をしたユーザインターフェイスの登録解除等の処理を記述

# Inventor APIの概要

## アドインのデバッグ

- プロジェクトのプロパティを確認し、デバッグの開始動作の外部プログラムの開始に Inventor.exeを指定
- F5キーでデバッグを開始



# 演習2

## Inventor アドイン

- Inventorがスタートした時に、Inventorのカレントのバージョンを見せるメッセージボックスを表示するアドインを作成してください

- 演習のヒント

- ✓ InventorのApplicationオブジェクトは、Activateメソッド内で、AddInSiteObject.Applicationで取得できます
- ✓ Inventorのバージョンは、SoftwareVersionオブジェクトのDisplayNameプロパティで取得できます
- ✓ SoftwareVersionオブジェクトは、ApplicationオブジェクトのSoftwareVersionプロパティから取得できます

# Inventor APIの概要

## 外部からのInventorの制御

- GetObject(Path, ProgID)またはCreateObject(ProgID)関数で、InventorのApplicationオブジェクトを取得
  - ✓ GetObject … 起動中のInventorのApplicationオブジェクトを取得
  - ✓ CreateObject…新しくInventorを起動し、起動したInventorのApplicationオブジェクトを取得
- InventorのProgIDは“Inventor.Application”
- .Netでは、System.Runtime.InteropServices.Marshal.GetActiveObject(GetObject関数に相当)およびSystem.Activator.CreateInstance(CreateObject 関数に相当)を使用

# Inventor APIの概要

## 外部からのInventorの制御

- VB.net

```
Dim _InvApplication As Inventor.Application = Nothing

Try
    Try ' Try to get an active instance of Inventor
        _InvApplication =
            System.Runtime.InteropServices.Marshal.GetActiveObject("I
nventor.Application")
    Catch ex As Exception
        End Try

    ' If not active, create a new Inventor session
    If _InvApplication Is Nothing Then

        Dim inventorAppType As Type =
System.Type.GetTypeFromProgID("Inventor.Application")
        _InvApplication = System.Activator.CreateInstance(inventorAppType)

        'Must be set visible explicitly
        _InvApplication.Visible = True

    End If

End Try
```

- C#

```
string progId = "Inventor.Application";
Type inventorApplicationType = Type.GetTypeFromProgID(progId);

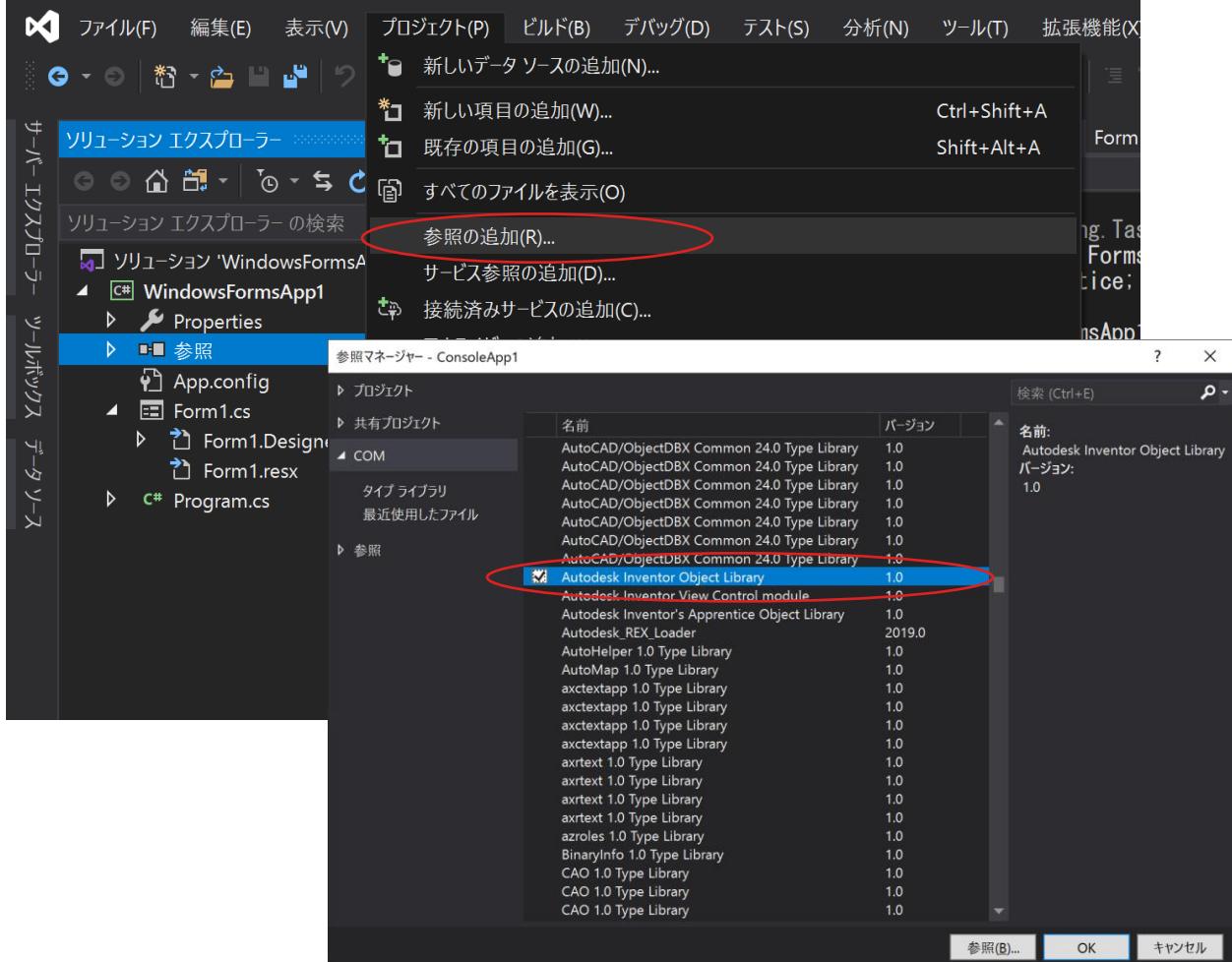
Inventor.Application instance = null;

try
{
    instance = (Inventor.Application)Marshal.GetActiveObject(progId);
    instance.Visible = true;
}
catch (COMException)
{
}

// No running instance, so proceed with creating a new instance
if (null == instance){
    instance =
    (Inventor.Application)Activator.CreateInstance(inventorApplicationType);
    instance.Visible = true;
}
```

# Inventor APIの概要

## Inventor Objectへの参照設定



- Visual Studio プロジェクトの参照の追加から、COMを選択し、Autodesk Inventor Object Libraryを追加

# 演習3

## 外部アプリケーションからの操作

- Visual StudioのWindowsフォームアプリケーションで、InventorにOpenされてアクティブになっているファイルの、ファイル名を表示するコードを作成してください

- 演習のヒント

- ✓ 起動中のInventorのApplicationオブジェクトは、GetObjectで、取得できます
- ✓ アクティブなドキュメントのオブジェクトは、ApplicationオブジェクトのActiveDocumentプロパティで取得できます
- ✓ ドキュメントのファイル名はFullNameプロパティで取得できます

# Inventor APIの概要

## Apprentice(アペレンテス) Server

- Inventor Apprentice Serverは、インプロセスで動作するInventorのデータへのアクセスを提供するActiveXサーバ
- Inventor Apprentice Server自身はGUIを持たず、InventorアプリケーションAPIのサブセットを提供
- Inventorを必要とせず、スタンドアロンで使用可能
- 無料で提供(Inventor View 製品の一部としてインストール済み)
- アセンブリ構成、B-Rep、図面シート、ビューへのアクセスは、読み込み専用の提供
- iProperty、アトリビュート、ファイル参照へのアクセスはRead/Write提供

# Inventor APIの概要

## Apprentice Server利用上の注意

- VB.net

'アペレンティスの作成

```
Dim oApprentice As ApprenticeServerComponent  
oApprentice = New ApprenticeServerComponent
```

'ドキュメントを開く

```
Dim oDoc As ApprenticeServerDocument  
oDoc = oApprentice.Open("Temp Part.ipt")  
MessageBox.Show("Opened: " & oDoc.DisplayName)
```

- C#

//アペレンティスの作成

```
ApprenticeServerComponent oApprentice  
= new ApprenticeServerComponent();
```

//ドキュメントを開く

```
ApprenticeServerDocument oDoc  
= oApprentice.Open("Temp Part.ipt");  
MessageBox.Show("Opened: " + oDoc.DisplayName);
```

# 演習4

## Apprentice Server

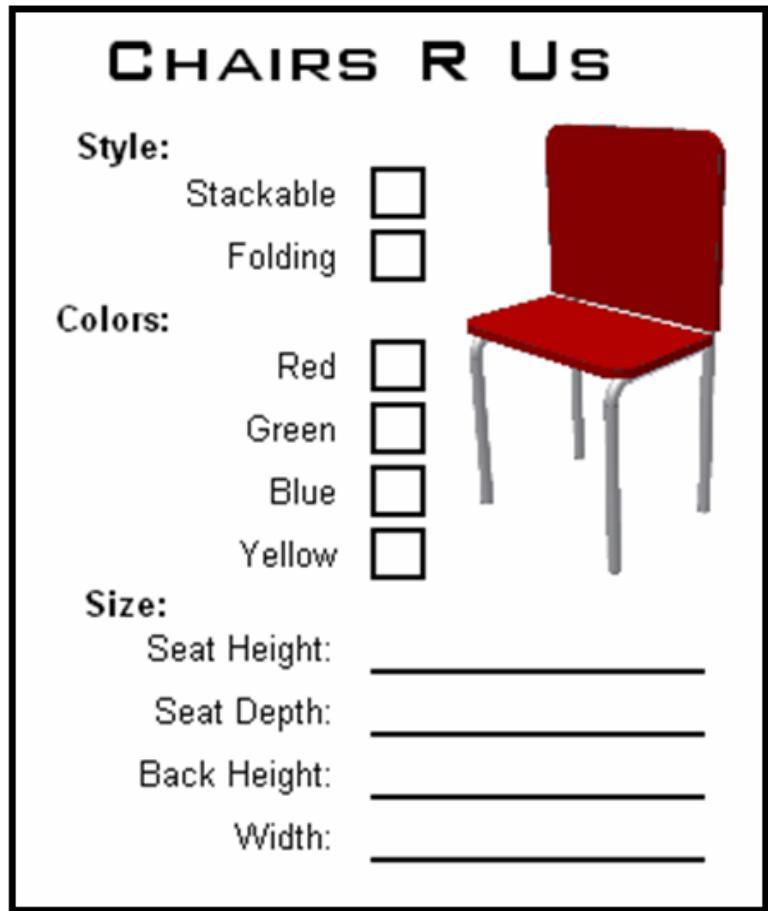
- Visual Studioの“Windows フォームアプリケーション”で、アペレンテスを使い、Inventor ファイルをOpenして、ファイル名を表示するコードを作成してください

# Inventor Object Model



# Inventor Object Model

## Object指向の基本的な用語

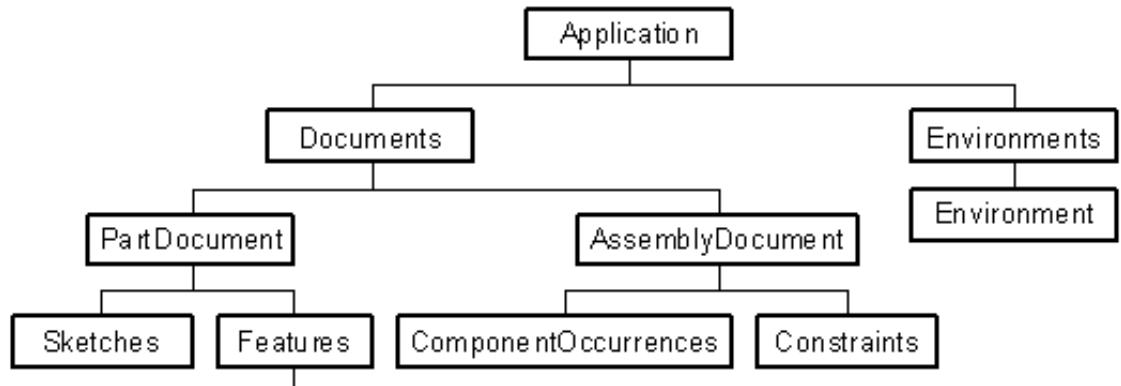


- オブジェクト…物を表す。左図の例では、椅子がオブジェクトとして扱われる
- プロパティ…オブジェクトの属性。左図の例では、Style、Color、Size等がプロパティとなる
- メソッド…オブジェクトが実行するアクション。左図の例では、移動、折りたたむ、廃棄する等がメソッドとなる
- イベント…オブジェクトに何かが発生した場合の通知
- クラス…オブジェクトの定義

# Inventor Object Model

## Inventor Object Modelの概要

### □ Inventor API Object Model



- InventorのAPIは、COM Automationの機能により Object 群として公開され、それぞれのオブジェクトがアプリケーション内で、いずれかと一致
- それぞれのオブジェクトはメソッド、プロパティ、イベントをサポート
- オブジェクトはObject Model を通してアクセスされる
- Object Modelの最上位のオブジェクトは Application オブジェクト。

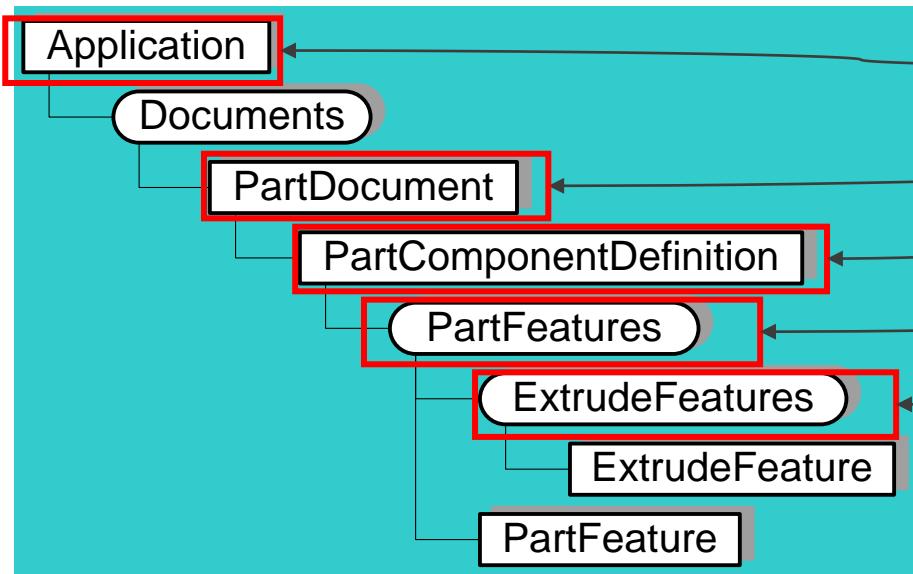
# Inventor Object Model

## Inventorオブジェクト

- オブジェクトは多くのメソッド・プロパティ・イベントをサポート。
- 幾つかのオブジェクトは、ベースクラスまたは、他の特定のオブジェクトのいずれかを使ったオブジェクトより派生。
- 全てのオブジェクトは、自身のオブジェクトタイプを表すType プロパティをサポート
- 多くのオブジェクトは、自身のオーナーに戻るためのParent プロパティをサポート

# Inventor Object Model

## Inventor Object Modelの例



```
Public Sub GetExtrudeFeature()

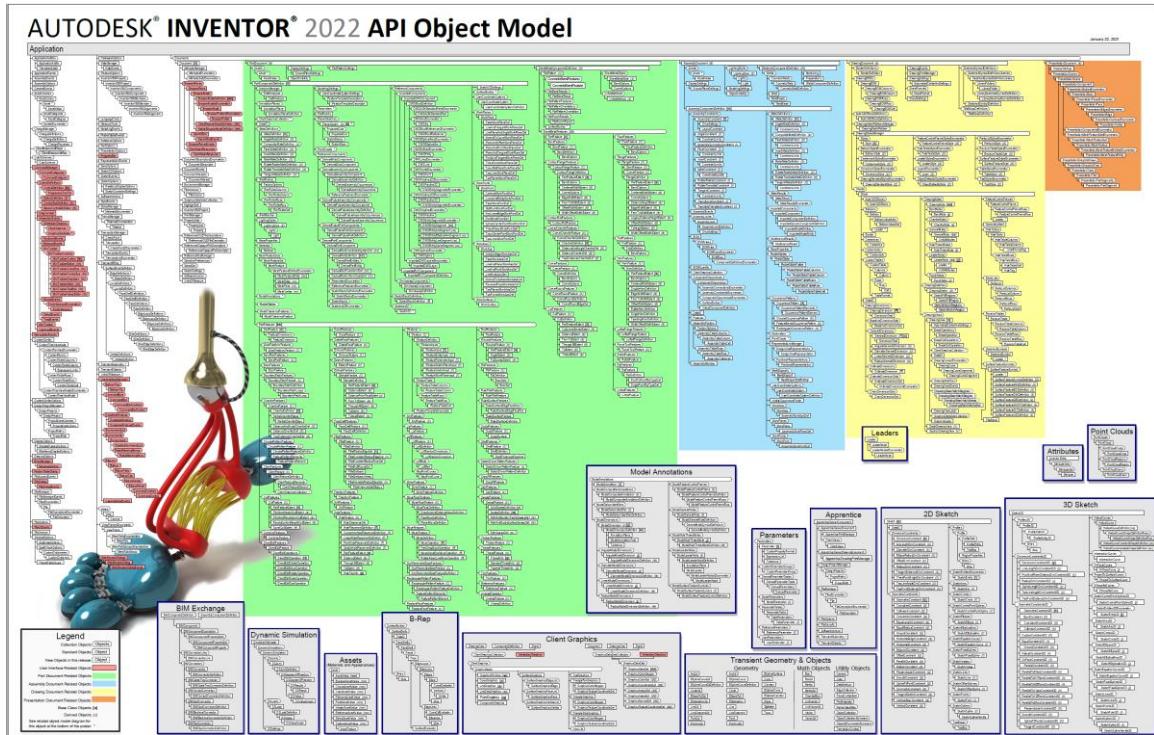
    Dim oPartDoc As PartDocument
    oPartDoc = ThisApplication.ActiveDocument

    Dim oExtrude As ExtrudeFeature
    oExtrude =
    oPartDoc.ComponentDefinition.Features.ExtrudeFeatures("Extrusio
    n1")
    MsgBox ("Extrusion " & oExtrude.Name & " is suppressed: " &
    oExtrude.Suppressed)

End Sub
```

# Inventor Object Model

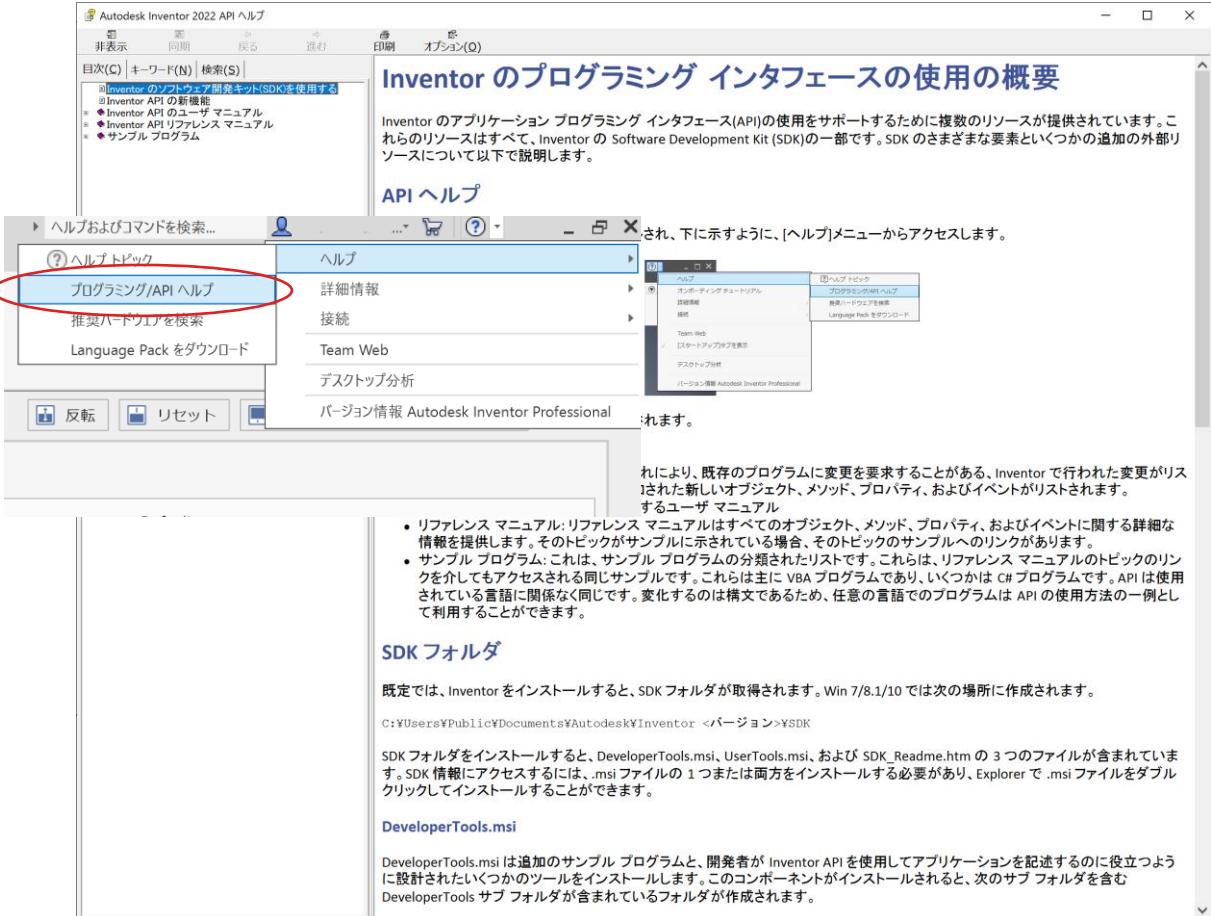
## Inventor Object Model 構造図



- C:\Users\Public\Documents\Autodesk\Inventor <version>\SDK 配下の DeveloperTools.msiをインストール後に、C:\Users\Public\Documents\Autodesk\Inventor <version>\SDK\DeveloperTools\Docs 配下に展開される Inventor<version>ObjectModel.pdfファイル

# Inventor Object Model

## Inventor APIリファレンス



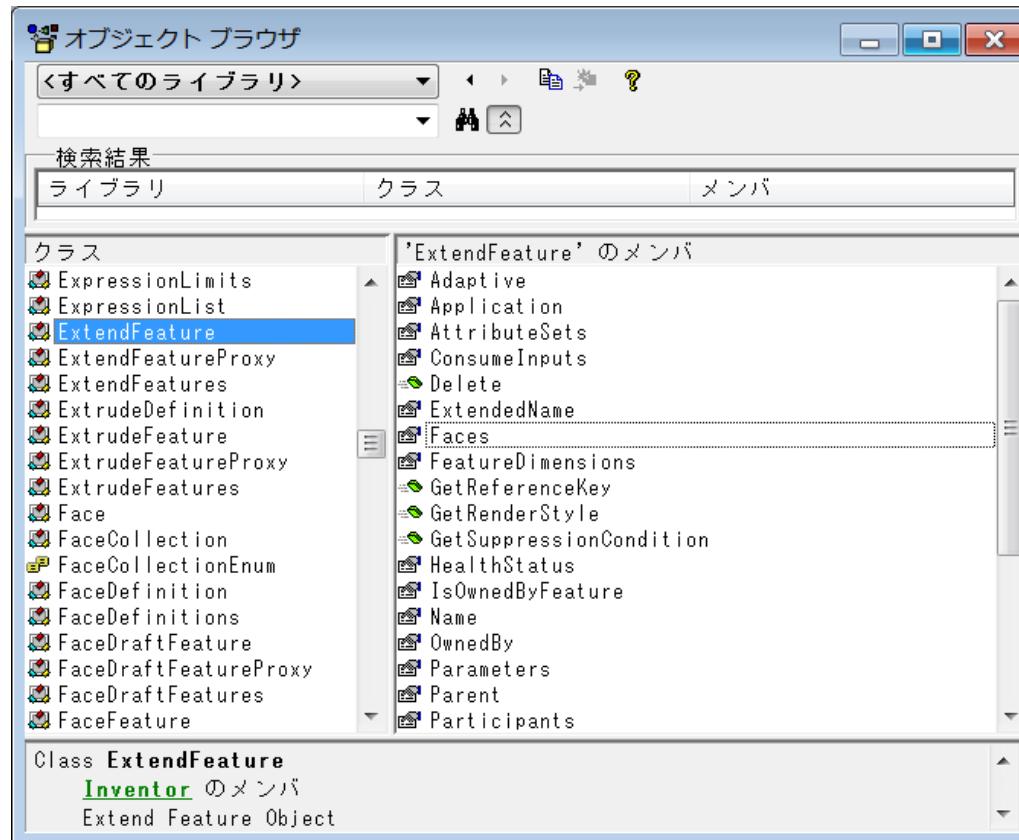
- Inventorのヘルプメニューの”プログラミング/API ヘルプ”からアクセス
- 日本語版ヘルプは以下の記事中から、ダウンロード  
[https://adndevblog.typepad.com/technology\\_perspective/2021/04/japanese-inventor-2022-programming-api-help.html](https://adndevblog.typepad.com/technology_perspective/2021/04/japanese-inventor-2022-programming-api-help.html)

# Inventor Object Model

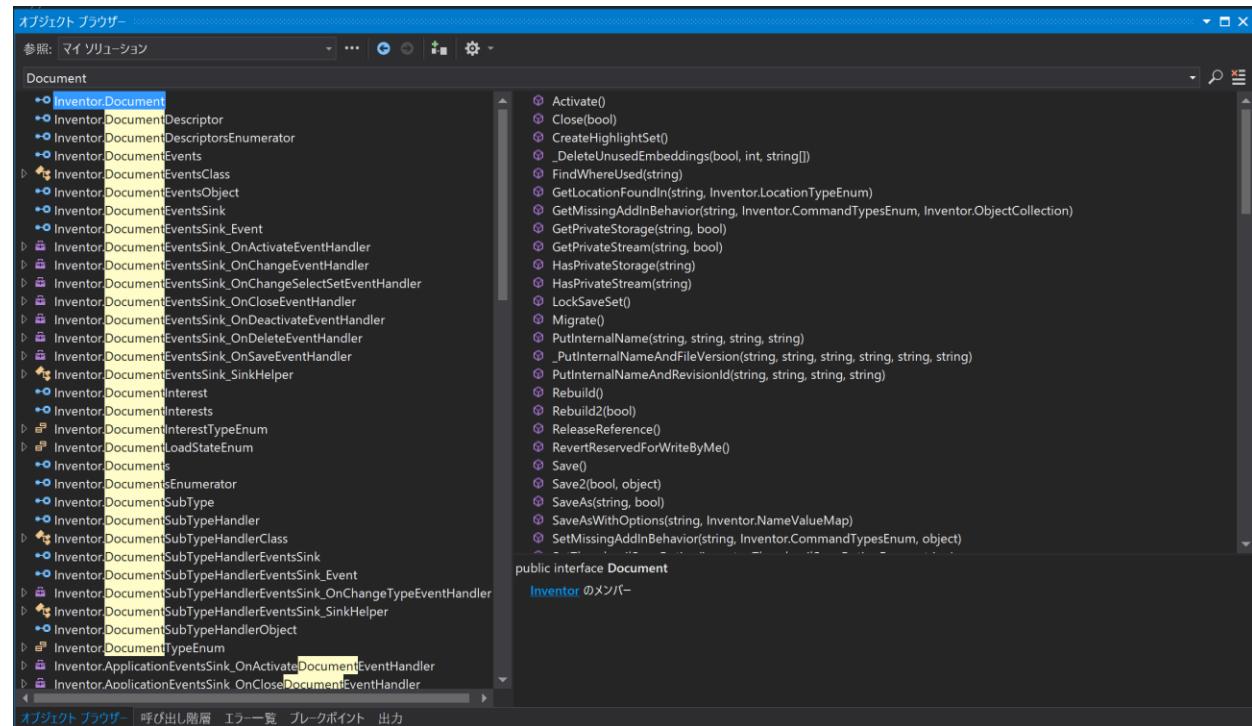
## オブジェクトブラウザ

- オブジェクトのプロパティ、メソッド、イベントを参照するツール

□ VBA IDE (F2キーまたは表示メニューまたは、ツールバーの  ボタン)



□ Visual Studio IDE ( Ctrl + W,Jキーまたは表示メニュー)

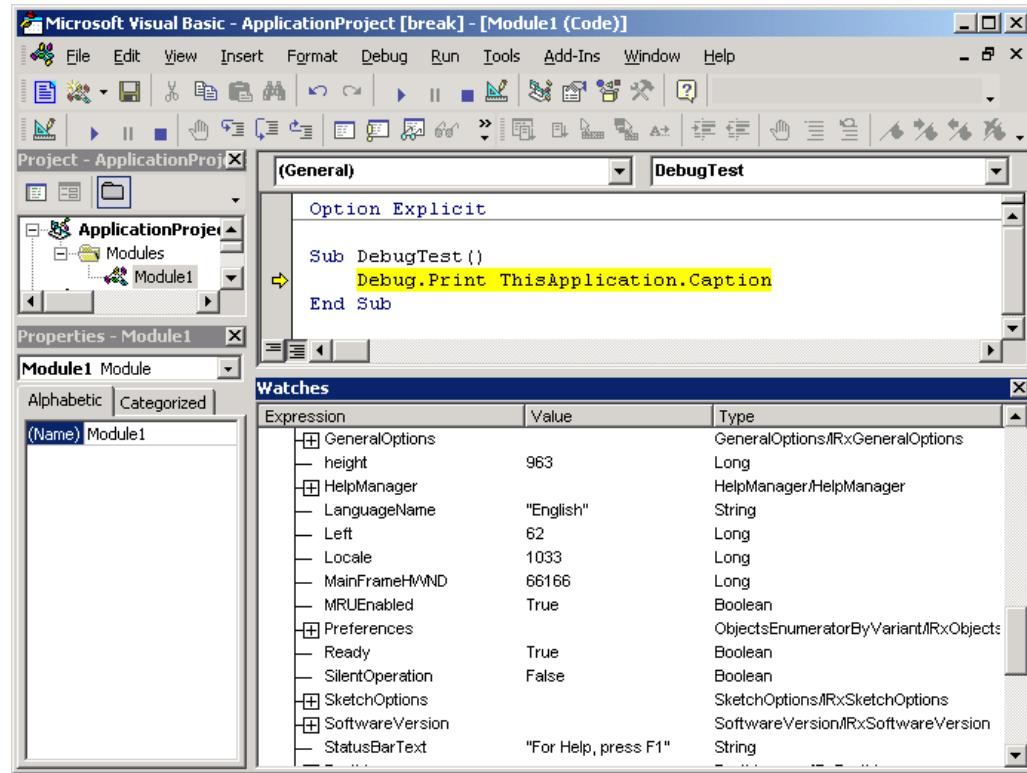


# Inventor Object Model

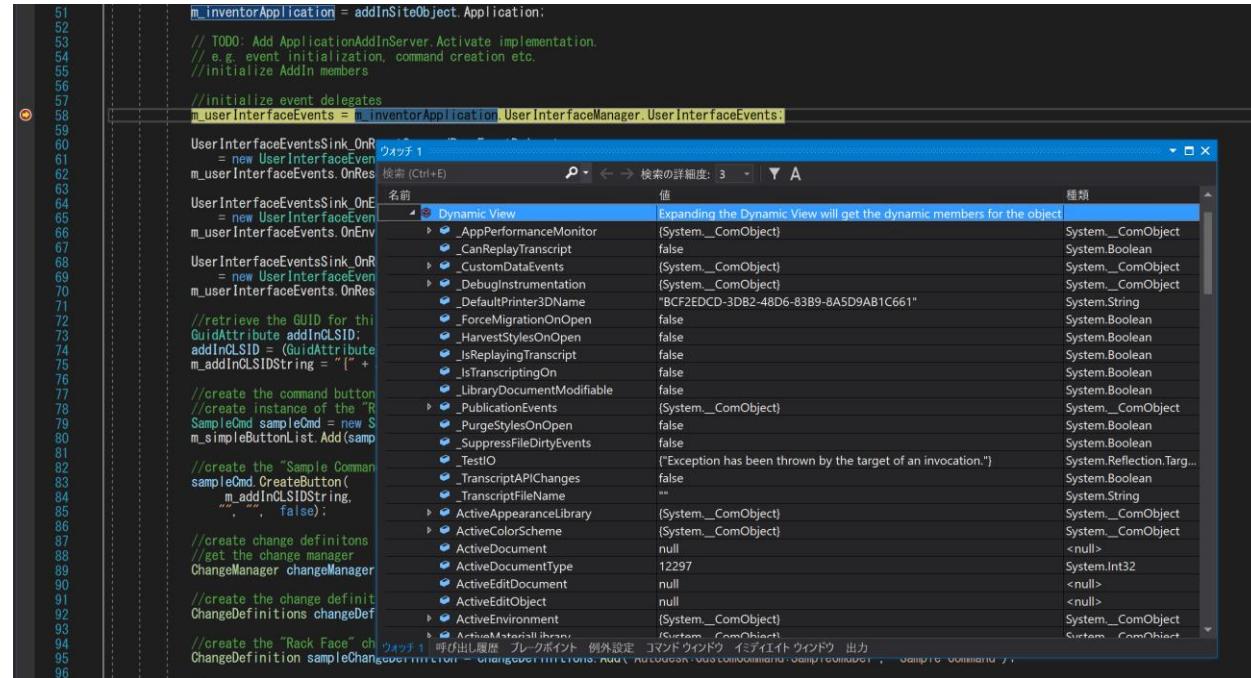
## デバッガ

- 実データでオブジェクトモデルを表示。オブジェクトのプロパティ値やコレクションの要素を参照可能

### □ VBA IDE



### □ Visual Studio IDE



# Inventor Object Model

## Collection/Enumeratorオブジェクト

- オブジェクトのリストへのアクセス提供
- コレクション内にオブジェクトがいくつあるかを示すCount プロパティのサポート
- Item プロパティで、コレクション内の特定のオブジェクトを取得(dotNet 環境で Item プロパティの実装が無いコレクションはコレクションの配列 [ ] で直接アクセス
  - ✓ Itemプロパティは(値:ベースインデックス =1 )でインデックスをサポート
  - ✓ 幾つかはのコレクションでは、オブジェクトの名前(文字列)でのアクセスをサポート
- ほとんどのコレクションはAdd メソッドで新しくオブジェクトの追加をサポート。
- Enumeratorも同様にコレクションをだが、Count、Itemプロパティのみをサポートし、新規追加もできない

# Inventor Object Model

## Collection要素へのアクセス

- Count、Itemプロパティを使用

```
Dim oExtrude As ExtrudeFeature  
Dim i As Long  
  
For i = 1 To oExtrudeFeatures.Count  
    oExtrude = oExtrudeFeatures.Item(i)  
    Debug.Print(oExtrude.Name)
```

[Next](#)

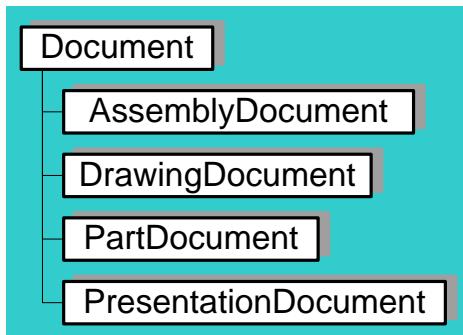
- For Eachステートメント

```
Dim oExtrude As ExtrudeFeature  
  
For Each oExtrude In oExtrudeFeatures  
    Debug.Print(oExtrude.Name)  
  
Next
```

# Inventor Object Model

## 派生オブジェクト

- 派生オブジェクトは、共通の親クラスを持つオブジェクト
- 同じ親クラスを持つオブジェクトは、親クラスの特性を共有する
- 以下の例では、ActiveDocumentプロパティで取得できるドキュメントは、Assembly、Part、Drawing、Presentationのいずれかとなるが、親クラスのDocumentとして取得し、すべてのDocumentに共通なSave()メソッドを実行



```
Public Sub SaveDoc()  
  
    'Get ActiveDocument, could be Part, Assembly or Drawing  
    Dim oDoc As Document  
    oDoc = ThisApplication.ActiveDocument  
  
    'Call the Save method on the generic 'Document' object  
    oDoc.Save()  
  
End Sub
```

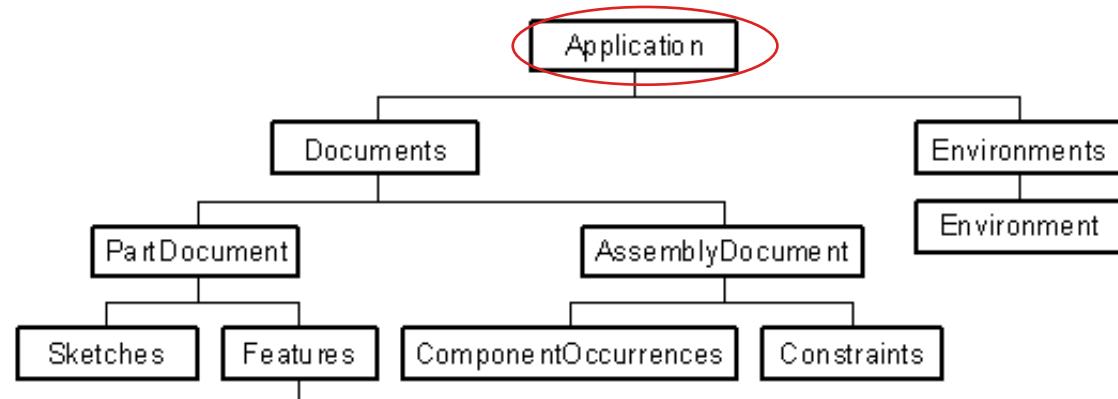


# Application オブジェクト

# Application オブジェクト

## Inventor Object Modelの概要

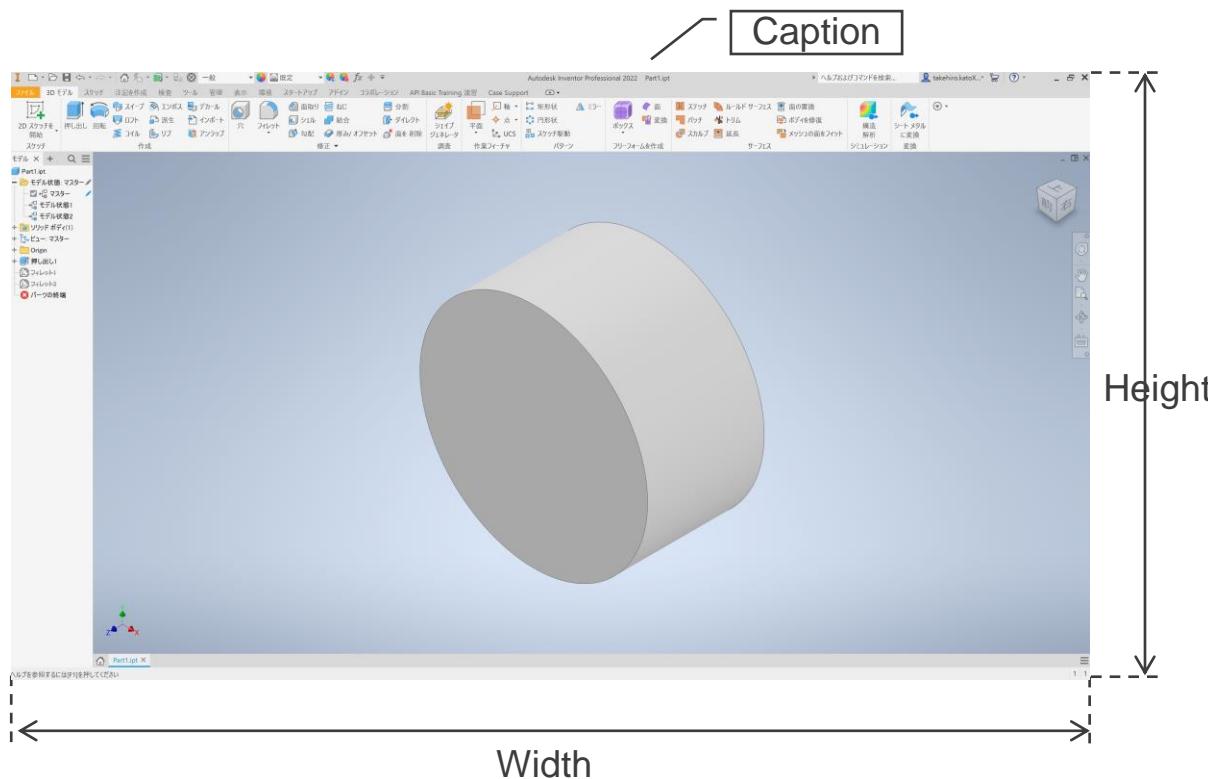
- Inventor Object Modelの最上位のオブジェクト



- Inventorのアプリケーションに対応するオブジェクト
- Applicationオブジェクトを通じて、他のInventorオブジェクトへのアクセスを提供
- Inventorドキュメントに依存しない、全般的な機能を提供
- イベント通知

# Application オブジェクト

## Applicationウィンドウ



■ アプリケーションオブジェクトは、様々なメソッドとプロパティを通してメインウィンドウへのアクセスを提供

- ✓ Caption
- ✓ Left, Top, Width, Height, GetAppFrameExtents, Move
- ✓ WindowState
- ✓ Visible
- ✓ MainFrameHwnd

# Application オブジェクト

## Utility オブジェクト

- SoftwareVersion … Inventor のバージョンに関する情報の提供

SoftwareVersion	
Properties	
BetaVersion	
BuildIdentifier	
DisplayName	
DisplayVersion	
Is64BitVersion	
Major	
Minor	
NotProduction	
ProductEdition	
ServicePack	
Type	

- TransientGeometry … 一時的なジオメトリオブジェクト

TransientGeometry	
Properties	Methods
PointTolerance	
Type	
	CreateArc2d CreateArc2dByThreePoints CreateArc3d CreateArc3dByThreePoints CreateBSplineCurve CreateBSplineCurve2d CreateBSplineCurve2dDefinition CreateBSplineCurveDefinition CreateBSplineSurface CreateBox CreateBox2d CreateCircle CreateCircle2d CreateCircle2dByThreePoints CreateCircleByThreePoints
	CreateCone CreateCylinder CreateEllipseFull CreateEllipseFull2d CreateEllipticalArc CreateEllipticalArc2d CreateEllipticalCone CreateEllipticalCylinder CreateFittedBSplineCurve CreateFittedBSplineCurve2d CreateLine CreateLine2d CreateLineSegment CreateLineSegment2d CreateMatrix
	CreateMatrix2d CreatePlane CreatePlaneByThreePoints CreatePoint CreatePoint2d CreatePolyline2d CreatePolyline2dFromCurve CreatePolyline3d CreatePolyline3dFromCurve CreateSphere CreateTorus CreateUnitVector CreateUnitVector2d CreateVector CreateVector2d

- TransientObjects … 一時的なユーティリティオブジェクト

TransientObjects	
Methods	
CreateColor	
CreateDataMedium	
CreateEdgeCollection	
CreateFaceCollection	
CreateFileMetadata	
CreateNameValueMap	
CreateObjectCollection	
CreateObjectCollectionByVariant	
CreateSignatureString	
CreateTranslationContext	

- ChangeManager、CommandManager、FileManager、HelpManager、TransactionManager、MeasureTools、UserInterfaceManager … 特定領域に関する機能へのアクセスの提供

# Application オブジェクト

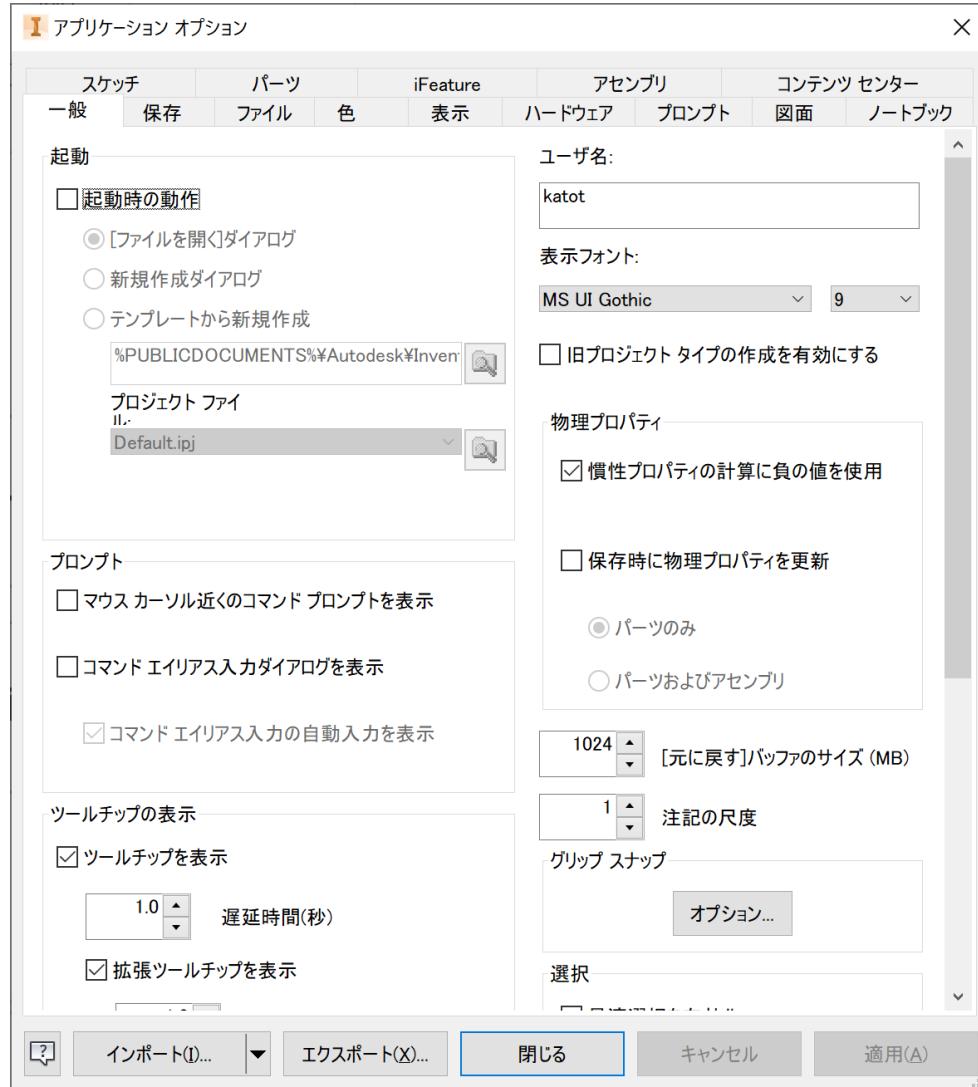
## Active オブジェクトへのショートカット

- 様々な“Active”オブジェクトへの直接アクセスを提供するプロパティ

- ActiveColorScheme
- ActiveDocument
- ActiveView
- ActiveEnvironment
- ActiveEditObject … 現在編集中のオブジェクトを戻す。これは、オープン中のスケッチ、シート、フラットパターン等のドキュメントを含む
- ActiveEditDocument … 現在編集中のドキュメントを戻す

# Application オブジェクト

## Application Option



■様々なアプリケーションオプションオブジェクトへのアクセスを提供

- AssemblyOptions
- ContentCenterOptions
- DisplayOptions
- DrawingOptions
- FileOptions
- GeneralOptions
- HardwareOptions
- iFeatureOptions
- NotebookOptions
- PartOptions
- PresentationOptions
- SaveOptions
- Sketch3DOptions
- SketchOptions
- ColorSchemes

プロンプトのタブは未実装

# Application オブジェクト

## その他のオブジェクト

- Documents ……全ての開かれているドキュメントと他のドキュメントにより参照されているものを含める
- ApplicationAddIns ……利用可能なアドイン
- VBAProjects ……VBEVBA 関連オブジェクト

# Application オブジェクト

## イベント

- 複数の異なるイベントセットへのアクセスの提供
  - ✓ ApplicationEvents
  - ✓ AssemblyEvents
  - ✓ FileAccessEvents
  - ✓ FileUIEvents
  - ✓ ModelingEvents
  - ✓ RepresentationEvents
  - ✓ SketchEvents
  - ✓ StyleEvents

# Application オブジェクト

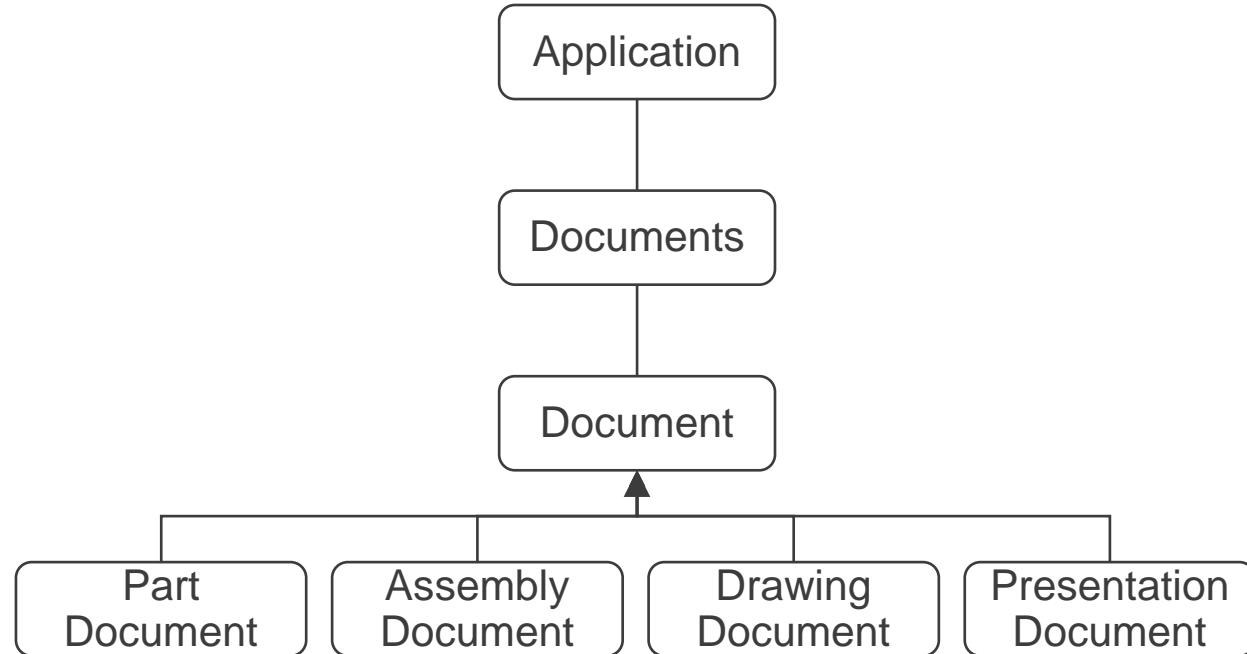
## その他

- SilentOperation…ダイアログ表示を抑制して、ディフォルト動作をする。バッチ処理操作で、ファイルオープン、処理、クローズ時に通常表示される警告ダイアログを抑止するのに役立つ
- LanguageName、Locale…言語情報
- MRUEnabled…ファイルメニューの一番下に、最近使われたファイルの表示の制御
- Ready…Inventor が完全に初期化されたかを示す
- StatusBarText…ステータスバーに示されるテキスト

# Documentオブジェクトの概要

# Documentオブジェクトの概要

## Document Type



- Inventorは、異なるデータタイプのドキュメントを持っており、それぞれに対応するInventorオブジェクトがある
  - ✓ パーツドキュメント (\*.ipt)
  - ✓ アセンブリドキュメント (\*.iam)
  - ✓ 図面ドキュメント (\*.idw)
  - ✓ プrezンテーションドキュメント (\*.ipn)
- Documentオブジェクトは、それら異なる種類の親クラスとなり、ドキュメントの種類に依存しないメソッドやプロパティを提供

# Documentオブジェクトの概要

## Documentへのアクセス



- 新規ドキュメントの作成

- ✓ Documents. Add

- 既存のドキュメントを開く

- ✓ Documents. Open

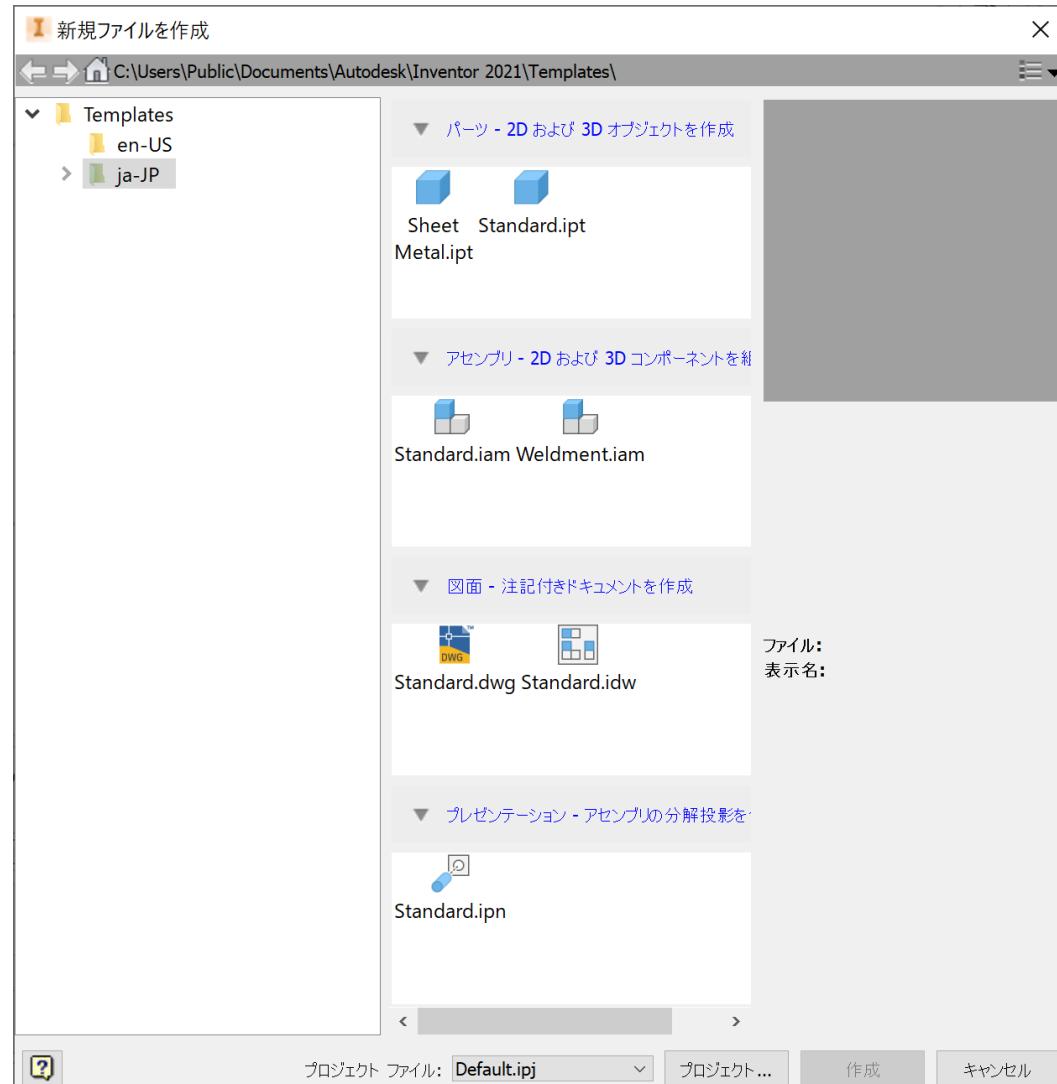
- 開いたドキュメントのアクセス

- ✓ Documents. Item

(Documentsで取得されるEnumerateは、開かれている全てのドキュメントと、そこから参照されているドキュメントを含む)

# Documentオブジェクトの概要

## Template



■ FileManagerオブジェクトの GetTemplateFile メソッドで指定されたテンプレートのフルパスを取得

- ✓ GetTemplateFile(  
    DocumentType As DocumentTypeEnum,  
    [SystemOfMeasure As SystemOfMeasureEnum =  
        kDefaultSystemOfMeasure],  
    [DraftingStandard As DraftingStandardEnum =  
        kDefault\_DraftingStandard],  
    [DocumentSubType]) As String

# Documentオブジェクトの概要

## ドキュメントのオープンと新規作成サンプルコード(VB.net)

・既存のドキュメントをオープン

```
Public Sub OpenDoc()

    Dim oDoc As Document
    oDoc = _InvApplication.Documents.Open("C:\Temp\Part1.ipt")

End Sub
```

・テンプレートを指定して新規作成

```
Public Sub CreateDoc()

    Dim oDoc As PartDocument
    oDoc = _InvApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject,
                                         _InvApplication.FileManager.GetTemplateFile(DocumentTypeEnum.kPartDocumentObject),
                                         True)

End Sub
```

・内部の規定値のテンプレートを指定して新規作成

```
Public Sub CreateDoc2()

    Dim oDoc As PartDocument
    oDoc = _InvApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject, , True)

End Sub
```

# Documentオブジェクトの概要

## ドキュメントの保存サンプルコード(VB.net)

- 新規ドキュメントに対し、名前を付けて保存は、SaveCopyAs フラグを False にセットしSaveAs メソッドを使い保存

```
Dim oDoc As PartDocument  
oDoc = _InvApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject)  
oDoc.SaveAs("C:\Temp\SaveTest.ipt", False)
```

- 既存のドキュメントは、SaveメソッドまたはSaveCopyAsメソッドをTrueにセットしSaveAsメソッドを使い保存.

```
oDoc.Save()  
oDoc.SaveAs("C:\Temp\SaveTest2.ipt", True)
```

# Documentオブジェクトの概要

## ドキュメントを閉じる

- Document.Close([SkipSave As Boolean = False])

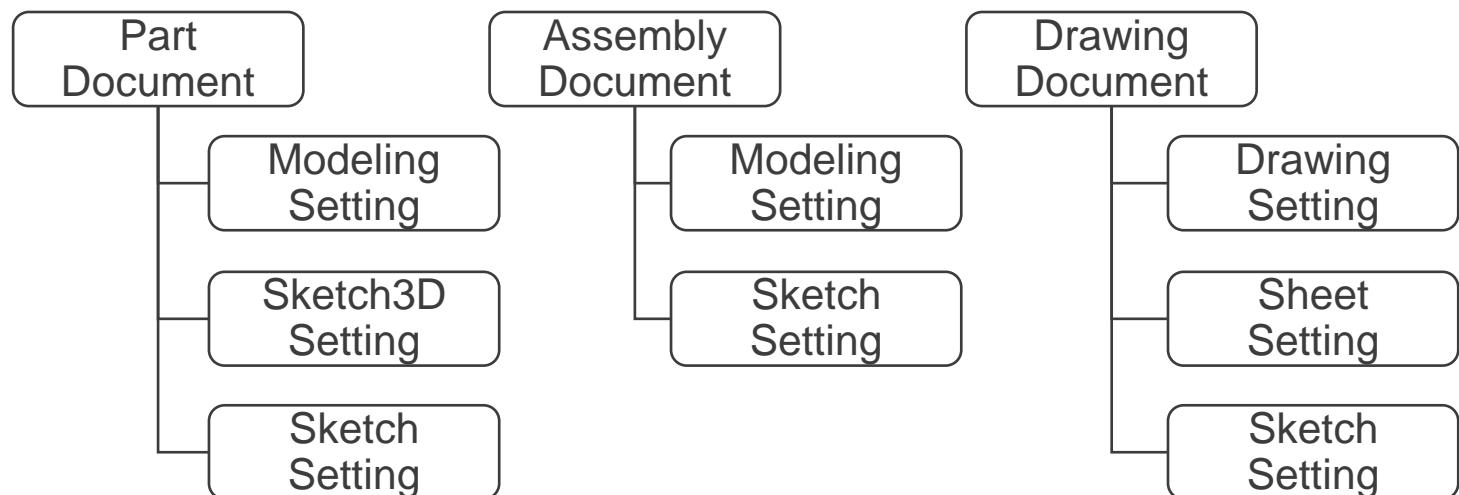
SkipSave 引数を使うことで、編集後のドキュメントを保存するかを確認するダイアログの表示を抑止し、編集を保存しないでドキュメントを閉じる事が可能。

# Documentオブジェクトの概要

## ドキュメント設定

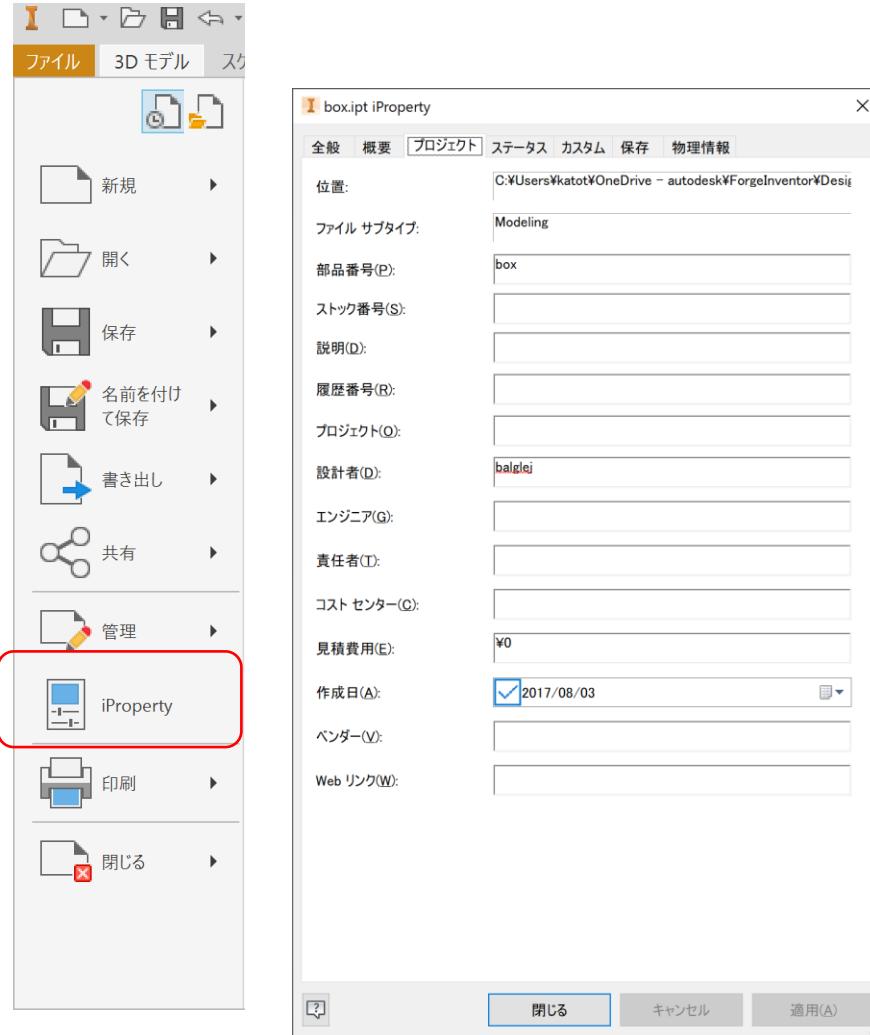


- Documentオブジェクトから取得できる様々なオブジェクトにより、ドキュメント設定にアクセス



# Documentオブジェクトの概要

## GUIからのiPropertyへのアクセス



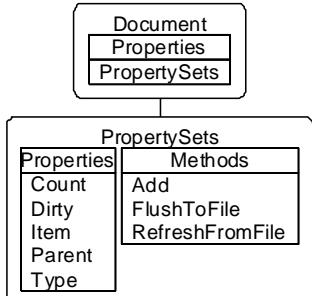
- iPropertyはドキュメントの関連情報の設定に使用
- 事前定義済みのプロパティセットは、iProperty ダイアログからアクセス可能
- iProperty ダイアログの“カスタム”タブから、ユーザー定義プロパティの追加作成が可能
- InventorとApprenticeの両方でサポート

# Documentオブジェクトの概要

## iPropertyのアクセス～ PropertySets、PropertySet、Propertyオブジェクト

### □ PropertySetsオブジェクト

PropertySetsコレクションは、全てのプロパティのコンテナとしての役割。Itemプロパティを使って、各々のPropertySetオブジェクトのアクセスを提供

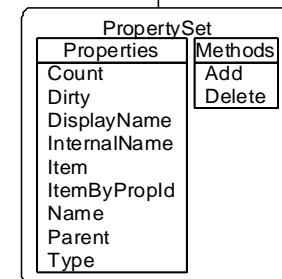


### □ PropertySetオブジェクト

PropertySetオブジェクトは、プロパティのコレクション。ほとんどのPropertySetオブジェクトは、iPropertyダイアログ内のタブの1つに対応。

PropertySetオブジェクトは以下の値で一意に特定される

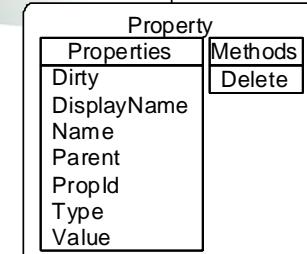
- ✓ InternalName (不变)
- ✓ Name (不变)
- ✓ DisplayName (変更の可能性)



### □ Property

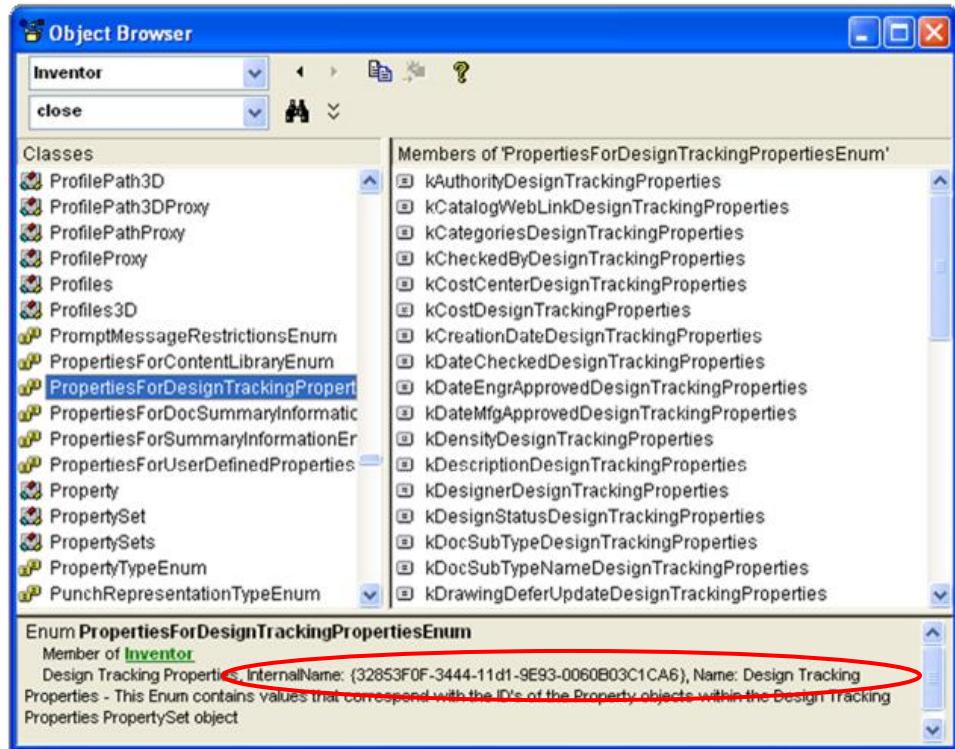
Propertyは、名前をつけられた値。Propertyオブジェクトは以下の値で一意に特定される

- ✓ ID (不变)
- ✓ Name (不变)
- ✓ DisplayName (変更の可能性)



# Documentオブジェクトの概要

## iPropertyのProperty値



- Property値は 内部的には、Variant値として保管され、以下の型をサポート
  - ✓ Integer, Long, Double, String, Date, Boolean(サムネイル画像のIPictureDispを除く)
- InternalNameはオブジェクトブラウザ、またはSDKのヘッダファイル(SDK\Include\PropFMTIDs.h)で確認が可能
- PropIdは、プロパティに関連したenumsで定義される

# Documentオブジェクトの概要

## iProperty PropertySet名

- Inventorの定義済みPropertySetsの名前と内部名
  - ✓ Inventor Summary Information (Inventor 概要情報)  
**{F29F85E0-4FF9-1068-AB91-08002B27B3D9}**
  - ✓ Inventor Document Summary Information (Inventor ドキュメント概要情報)  
**{D5CDD502-2E9C-101B-9397-08002B2CF9AE}**
  - ✓ Design Tracking Properties (デザイントラッキングプロパティ)  
**{32853F0F-3444-11D1-9E93-0060B03C1CA6}**
  - ✓ Inventor User Defined Properties (Inventor ユーザー定義プロパティ)  
**{D5CDD505-2E9C-101B-9397-08002B2CF9AE}**

# Documentオブジェクトの概要

## iProperty プロパティアクセスのサンプルコード(VB.Net)

```
Public Sub iPropAccess()

    Dim oDoc As Document
    oDoc = _InvApplication.ActiveDocument

    ' デザイントラッキングプロパティセットを取得
    Dim oDTProps As PropertySet
    oDTProps = oDoc.PropertySets.Item("{32853F0F-3444-11d1-9E93-0060B03C1CA6}")

    'ディスプレイ名または名前を使って同じプロパティセットへアクセス 表示名でのアクセスはローカライズに依存する
    'ため非推奨
    'oDTProps = oDoc.PropertySets.Item("Design Tracking Properties")

    ' PropIdを使用して、デザイナープロパティを取得
    Dim oDesignerProp As Inventor.Property
    oDesignerProp = oDTProps.ItemByPropId(
        PropertiesForDesignTrackingPropertiesEnum.kDesignerDesignTrackingProperties)

    ' 表示名を指定することも可能だが、変更される可能性がある
    ' oDesignerProp = oDTProps.Item("Designer")

    ' 表示名と値を表示
    Debug.Print(oDesignerProp.DisplayName & " = " & oDesignerProp.Value)

    ' プロパティ値を変更
    oDesignerProp.Value = "Bill & Ted"

End Sub
```

# Documentオブジェクトの概要

iProperty 全てのプロパティ名、PropIdを取得するサンプルコード(VB.Net)

```
Public Sub DumpDocProperties()

    Dim oDocument As Document
    oDocument = _InvApplication.ActiveDocument

    Dim oPropertySets As PropertySets
    oPropertySets = oDocument.PropertySets

    Dim oPropertySet As PropertySet

    'Namespace required to avoid conflict with Vb.Net "Property" keyword
    Dim oProperty As Inventor.Property

    For Each oPropertySet In oPropertySets

        Debug.Print(vbCrLf & "----- Property Set: " & oPropertySet.Name)

        For Each oProperty In oPropertySet

            Debug.Print(" - " & oProperty.Name & " [ID:" & oProperty.PropId & "] :" & Space(30 -
Len(oProperty.Name)) & oProperty.Expression)

        Next

    Next

End Sub
```

# Documentオブジェクトの概要

## iProperty 作成

- `PropertySets.Add (Name As String, [InternalName]) As PropertySet`
  - ✓ 名前と内部名は、ドキュメント内の他のプロパティセットに対して一意でなくてはならない
- `PropertySet.Add (PropValue, [Name], [PropId]) As Property`
  - ✓ プロパティは、カスタムプロパティへのセットを除いて、予め定義されたセットに追加はできない
  - ✓ 名前と PropId は、プロパティセット内の他のプロパティに対して一意でなくてはならない
  - ✓ 値のタイプは、配列とオブジェクトを除いた Variant タイプが可能
- `Property Sets`と `Property` は アンダースコアで始まる名前を使って、`Hidden` として作ることが可能。これらは、コレクションを通してインデックスで返されず、名前を使う事で、のみアクセスが可能

# Documentオブジェクトの概要

## iProperty 作成

- Inventorで、ドキュメントが保存される時にプロパティに加えられた変更は保存される
- Apprenticeでは、PropertySetsの FlushToFile メソッドを使用する必要がある

```
'Create Apprentice Application
Dim oApprentice As ApprenticeServerComponent
oApprentice = New ApprenticeServerComponent

'Open a document
Dim oDoc As ApprenticeServerDocument
oDoc = oApprentice.Open("C:\Temp\Part.ipt")

'change a property
Dim oPropSet As PropertySet
oPropSet = oDoc.PropertySets("Design Tracking Properties")

oPropSet.Item("Designer").Value = "Fred Astair"

'save the change
oDoc.PropertySets.FlushToFile()
```

# 演習5

## iProperty の作成と保存

以下のプログラムを作成してください

1. 新規パートドキュメントの作成
2. “作成者”プロパティの値を任意の名前に変更
3. 値が“Inventor トレーニングプロパティ”で、値がADN の、新しいカスタム（ユーザー定義）プロパティの作成
4. “C:\Temp\NewPart2.ipt”としてドキュメントの保存
5. ドキュメントのクローズ

### ■ 演習のヒント

- ✓ Inventorの定義済みPropertySets内部名  
Inventor 概要情報  
**{F29F85E0-4FF9-1068-AB91-08002B27B3D9}**
- Inventor ユーザー定義プロパティ  
**{D5CDD505-2E9C-101B-9397-08002B2CF9AE}**
- ✓ “作成者”プロパティのプロパティIdは以下の列挙値  
kAuthorSummaryInformation

# Documentオブジェクトの概要

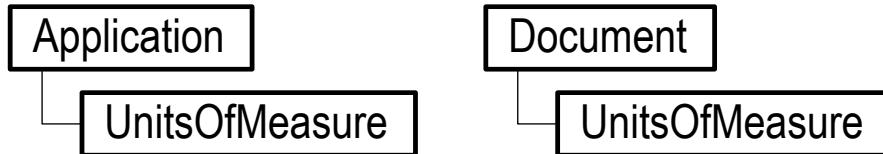
## 計測単位 UnitsOfMeasure オブジェクト



- 全てのInventor ドキュメントは同じ内部単位(データベース単位)を使う
  - ✓ Length: センチメートル
  - ✓ Angle: ラジアン
  - ✓ Time: 秒
  - ✓ Mass: キログラム
- ドキュメント設定ダイアログで、エンドユーザーが指示した単位に内部単位を変換
- UnitsOfMeasure オブジェクトは、APIを通して同様の機能を提供

# Documentオブジェクトの概要

## 計測単位 UnitsOfMeasure オブジェクト



- 単位変換に使用できるUnitsOfMeasureオブジェクトは、各ドキュメント及びApplicationオブジェクトから取得が可能
- ドキュメントから取得されるUnitsOfMeasureオブジェクトは、ドキュメント設定コマンドと等価な機能を提供
- UnitsOfMeasureオブジェクトは、プログラムから単位を取り扱うことを助ける機能を提供

UnitsOfMeasure	
Properties	Methods
AngleDisplayPrecision	CompatibleUnits
AngleUnits	ConvertUnits
LengthDisplayPrecision	GetDatabaseUnitsFromExpression
LengthUnits	GetDrivingParameters
MassUnits	GetLocaleCorrectedExpression
Parent	GetPreciseStringFromValue
TimeUnits	GetStringFromType
Type	GetStringFromValue
	GetTypeFromString
	GetValueFromExpression
	IsExpressionValid

# Documentオブジェクトの概要

## 計測単位 UnitsOfMeasure オブジェクトのUnit Type

- APIからのUnit Typeの指定は、2つの異なる方法で指定可能

1. UnitsTypeEnumで指定

- 特定の単位タイプを指定

例. kInchLengthUnits, kMillimeterLengthUnits, kDegreeAngleUnits, etc.

- エンドユーザーによって明示された、現在のディフォルトタイプ

例. kDefaultDisplayLengthUnits, kDefaultDisplayAngleUnits, etc.

- 内部ベース単位

例. kDatabaseLengthUnits, kDatabaseAngleUnits, etc.

2. 文字列で指定

例. “in”, “mm mm mm”, “m ^ 3”, “m /(s s)”

# Documentオブジェクトの概要

## 計測単位 UnitsOfMeasure 内部単位

- Inventorは、ユーザがドキュメントに設定した単位にかかわらず、内部的には同じ単位を使用
- ユーザが指定した精度にかかわらず、倍精度浮動小数点を使用

# Documentオブジェクトの概要

## 計測単位 ユーザ入力値チェックのサンプルコード(VB.net)

```
Private Sub TextBox1_Change()

    '長さとして妥当な表現かをチェック
    If Not m_oUOM.IsExpressionValid(TextBox1.Text, UnitsTypeEnum.kDefaultDisplayLengthUnits) Then
        ' 妥当な入力値ではない場合は、テキストを赤色に変更
        TextBox1.ForeColor = Drawing.Color.Red
    Else
        ' 妥当な入力値の場合は、テキストを黒色に変更
        TextBox1.ForeColor = Drawing.Color.Black
    End If

End Sub
```

# Documentオブジェクトの概要

## 計測単位 ユーザ入力値とスケッチラインの長さを比較 (VB.net)

```
' 入力文字を実数値として取得
Dim dValue As Double
dValue = m_oUOM.GetValueFromExpression(txtInput.Text,
                                         UnitsTypeEnum.kDefaultDisplayLengthUnits)

'スケッチラインの長さと比較
If System.Math.Abs(oSketchLine.Length - dValue) < 0.00001 Then
    MsgBox("ラインは入力値と同じ長さです")
Else
    MsgBox("ラインの長さは入力値とことなります")
End If
```

# Documentオブジェクトの概要

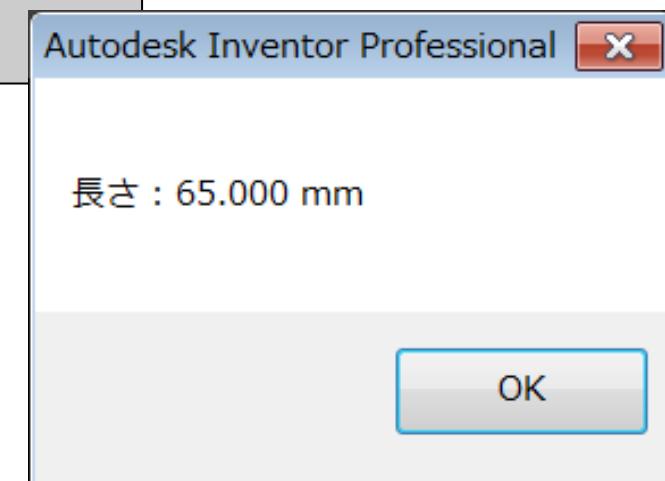
## 計測単位 値を表示 (VB.net)

```
Private Sub TestLength()
    ShowLength(6.5)
End Sub

Private Sub ShowLength(ByVal dLength As Double)

    ' 長さの表示文字列を取得
    Dim strLength As String
    strLength = m_oUOM.GetStringFromValue(dLength, _
        UnitsTypeEnum.kDefaultDisplayLengthUnits)

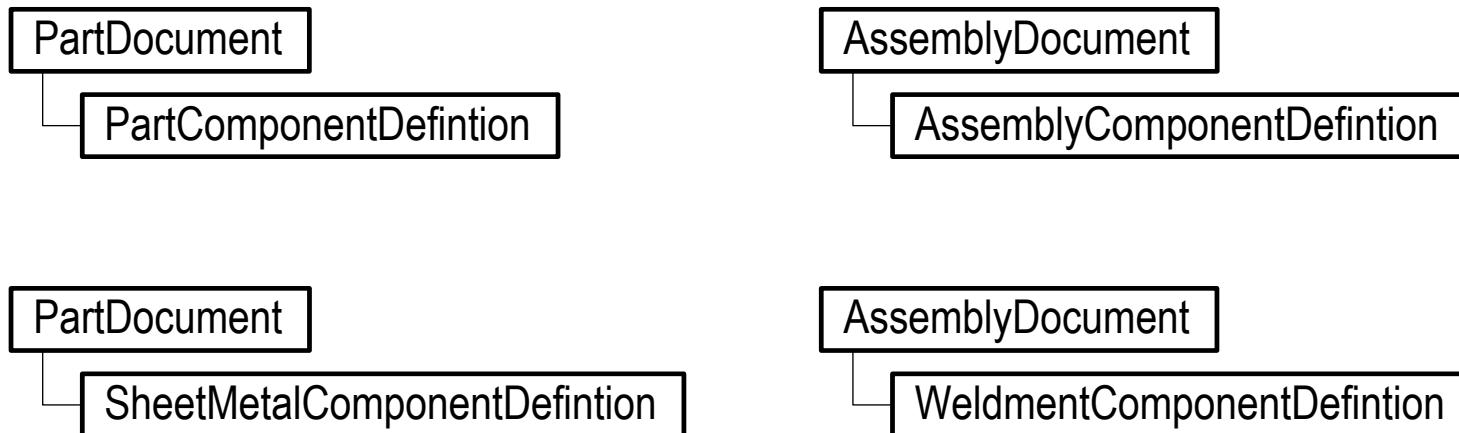
    MsgBox("長さ: " & strLength)
End Sub
```



# Documentオブジェクトの概要

## コンポーネント定義(Component Definition)

- パーツとアセンブリドキュメントでは、ComponentDefinition オブジェクトを通してパーツとアセンブリの定義情報にアクセス可能



- Drawingには、ComponentDefinitionは存在しない

# Documentオブジェクトの概要

## ユーザインターフェイスでのパラメータ

パラメータ

パラメータ名	使用者	単位/タイプ	計算式	表記値	駆動ルール	寸法公差	モデル値	キー	エクスポート
モデル パラメ...									
d0	Sketch1	in	length	24.000000		●	24.000000	<input type="checkbox"/>	<input type="checkbox"/>
d1	Sketch1	in	width	36.000000		●	36.000000	<input type="checkbox"/>	<input type="checkbox"/>
d2	Sketch1	in	thickness	0.500000		●	0.500000	<input type="checkbox"/>	<input type="checkbox"/>
d3	Sketch1	in	thickness	0.500000		●	0.500000	<input type="checkbox"/>	<input type="checkbox"/>
d4	Sketch1	in	thickness	0.500000		●	0.500000	<input type="checkbox"/>	<input type="checkbox"/>
d5	Sketch1	in	thickness	0.500000		●	0.500000	<input type="checkbox"/>	<input type="checkbox"/>
d6	Extrusion1	in	height	16.000000		●	16.000000	<input type="checkbox"/>	<input type="checkbox"/>
d7	Extrusion1	deg	0.0 deg	0.000000		●	0.000000	<input type="checkbox"/>	<input type="checkbox"/>
d8	Extrusion2	in	thickness	0.500000		●	0.500000	<input type="checkbox"/>	<input type="checkbox"/>
d9	Extrusion2	deg	0.0 deg	0.000000		●	0.000000	<input type="checkbox"/>	<input type="checkbox"/>
ユーザ パラメ...									

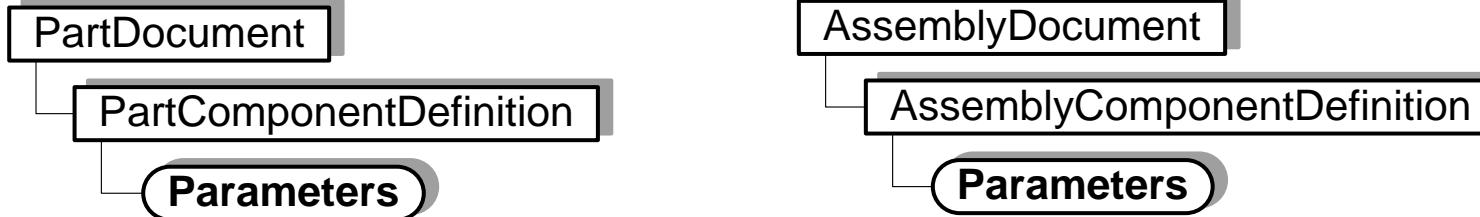
数値を追加 更新 未使用の項目を削除 公差をリセット << シンプル 完了

リンク  すぐに更新

?

# Documentオブジェクトの概要

## APIからのパラメータへのアクセス



```
' パラメータのコレクションを取得  
Dim oParameters As Parameters =  
_ InvApplication.ActiveDocument.ComponentDefinition.Parameters
```

# Documentオブジェクトの概要

## パラメータ GUIとAPIの対応

パラメータ名	Parameter.Name
High	例: Parameter.Name = "Length"
単位	Parameter.Units
mm	例: Parameter.Units = "mm"
計算式	Parameter.Expression
Length / 2 ul	例: Parameter.Expression = "Hight/2 + 5 mm"
表記値	Parameter.Value
47.500000	例: MsgBox "Normal Value: " & Parameter.Value & " cm"
寸法公差	Parameter.ModelValueType
●	例: Parameter.ModelValueType = "kUpperValue"
モデル値	Parameter.ModelValue
47.500000	例: MsgBox "Model Value: " & Parameter.ModelValue & " cm"
	Parameter.ExposedAsProperty
	例: Parameter.ExposedAsProperty = "True"
コメント	Parameter.Comment
	例: Parameter.Comment = "コメントです"

# Documentオブジェクトの概要

## パラメータ値の設定サンプルコード(VB.net)

```
Public Sub SetParameter()

    ' パラメータオブジェクトを取得。アセンブリドキュメントがActiveであることを想定
    Dim oParameters As Parameters
    oParameters = _InvApplication.ActiveDocument.ComponentDefinition.Parameters

    ' パラメータ名が"Length"のパラメータを取得
    Dim oLengthParam As Parameter
    oLengthParam = oParameters.Item("Length")

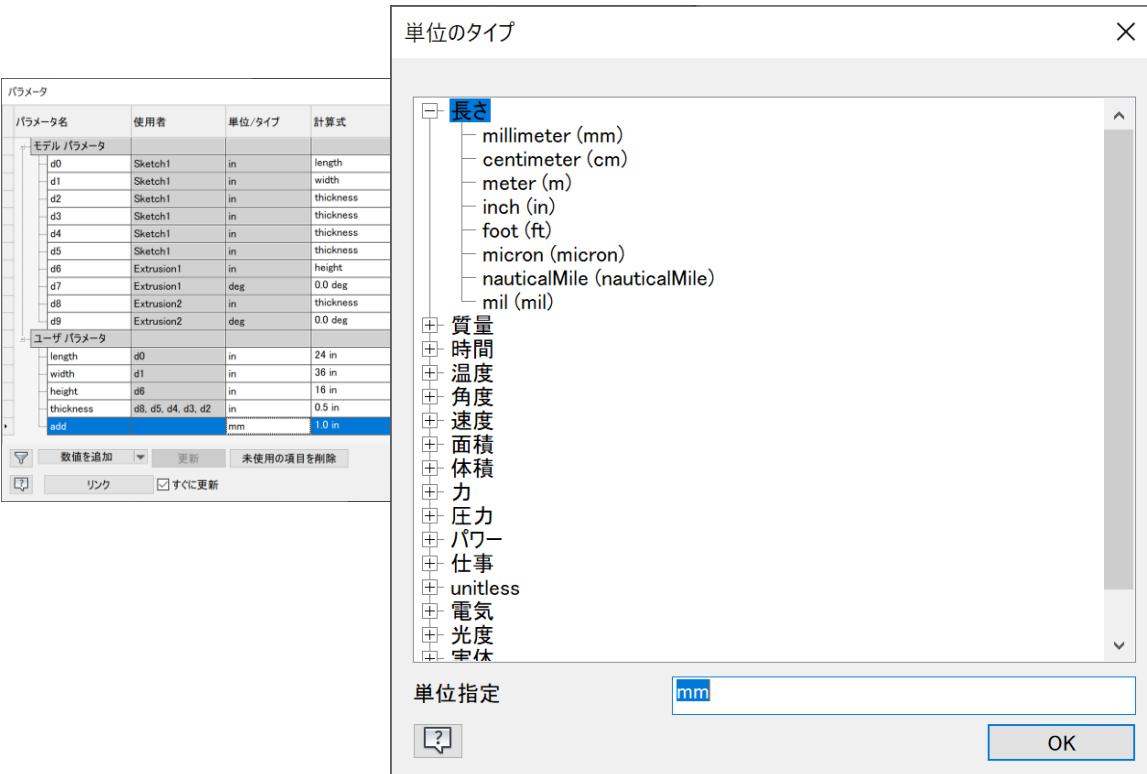
    ' パラメータを変更
    oLengthParam.Expression = "100 mm"

    ' ドキュメントの更新
    _InvApplication.ActiveDocument.Update()

End Sub
```

# Documentオブジェクトの概要

## パラメータの単位



- パラメータのUnitプロパティは、文字列または、UnitsTypeEnum値を使って設定が可能
- UnitsTypeEnumは、単位のタイプダイアログに表示されている、あらかじめ定義された単位に相当
- Unitプロパティに文字列で設定をする事は、単位のタイプダイアログで、文字列で単位指定することと同じであり、ダイアログと同様に既存の単位を組み合わせてのカスタム単位タイプを定義可能

# Documentオブジェクトの概要

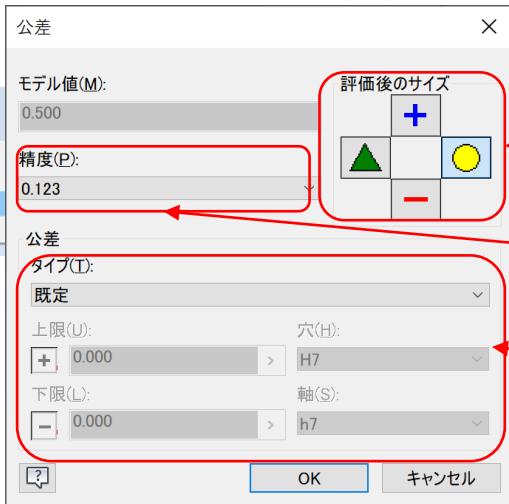
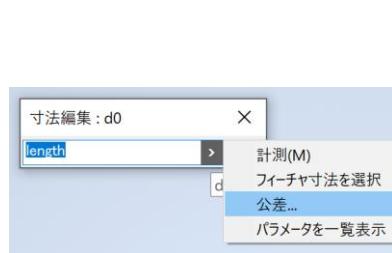
## パラメータの値

- ユーザーインターフェースとは異なり、パラメータの値を直接セット可能
- Valueプロパティはパラメータの実際の値をセットし、既存の数式も上書きする
- APIではパラメータ値は、**常に内部単位を使って定義**される
  - ✓ 長さ: Centimeters(センチメートル)
  - ✓ 角度: Radians (ラジアン)
    - $\pi \text{ radians} = 180^\circ$
    - $\pi = \text{Atn}(1) * 4$

# Documentオブジェクトの概要

## 公差(Tolerances)

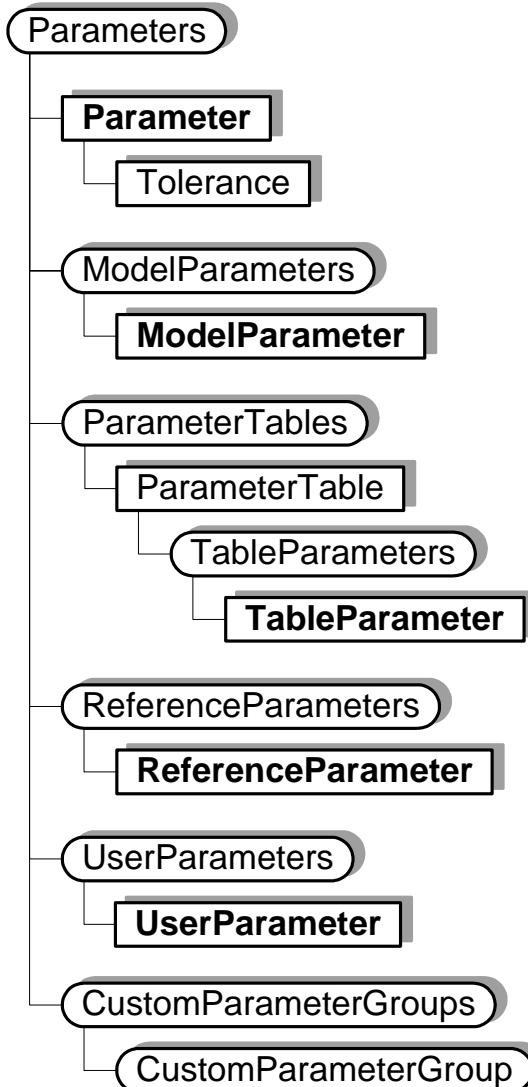
Parameter  
Tolerance



- 公差はモデルパラメータに定義が可能
- 公差オブジェクトは“公差”ダイアログと等しい機能を提供
- ModelValueType プロパティは、モデル値を計算するときにどの公差を使うか設定する
- Precisionプロパティは小数点の表示精度を設定する
- パラメータオブジェクトのSetToDefault・SetToDeviation・SetToFits・SetToLimits・SetToMax・SetToMin・SetToSymmetric メソッドは公差を定義。
  - ✓ Call oParam.Tolerance.SetToDeviation("0.125 in", "-0.0625 in")
  - ✓ Call oParam.Tolerance.SetToDeviation(2.54 / 8, -2.54 / 16)
- パラメータオブジェクトのSetAllToMax・SetAllToMedian・SetAllToMin・SetAllToNominal メソッドは、パラメータダイアログの“公差をリセット”相当機能を提供

# Documentオブジェクトの概要

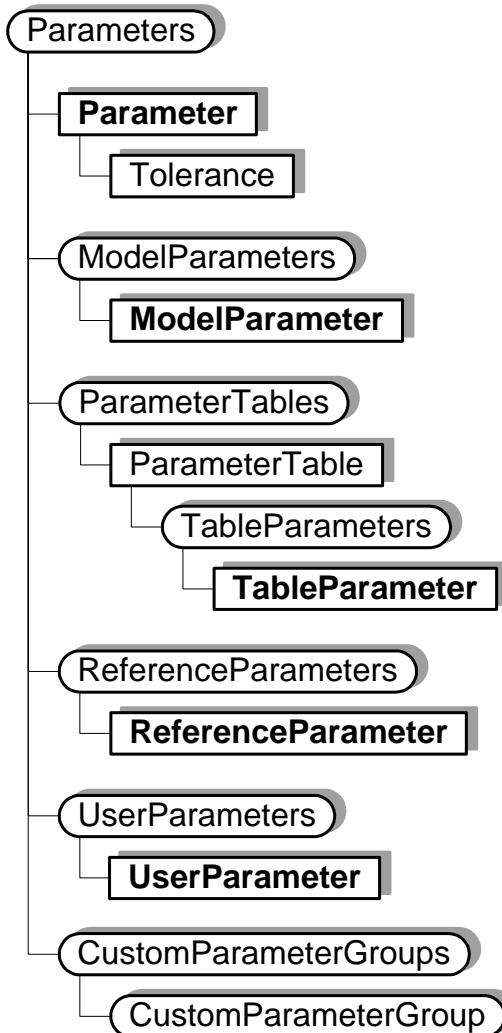
## パラメータタイプ



- Parametersコレクションは、全てのパラメータタイプのパラメータを返す
- ModelParameters・ParameterTables・ReferenceParameters・UserParameters オブジェクトは、特定のタイプのパラメータへのアクセスを提供

# Documentオブジェクトの概要

## パラメータの作成



- パラメータは、作成したいパラメータタイプのコレクションでメソッドを使うことによって作成可能
- ユーザーインターフェースでは、ユーザーパラメータのみが作成可能。他の全てのパラメータは操作の結果として間接的に作成される
- APIでは、ユーザー・モデル・参照パラメータを直接作成可能
- TableParametersはExcelワークシートのインポートによって作成される
- パラメータはAddByExpressionもしくはAddByValueメソッドを使って作成される

# Documentオブジェクトの概要

## パラメータの作成 サンプル(VB.net)

、ユーザパラメターの追加

```
Dim oUserParams As UserParameters
oUserParams = oCompDef.Parameters.UserParameters

Dim oParam As Parameter
oParam = oUserParams.AddByExpression("NewParam1", "3", _
    UnitsTypeEnum.kInchLengthUnits)

oParam = oUserParams.AddByExpression("NewParam1", "3", "in")

oParam = oUserParams.AddByExpression("NewParam2", "3", _
    UnitsTypeEnum.kDefaultDisplayLengthUnits)

oParam = oUserParams.AddByExpression("NewParam2", "3 in", _
    UnitsTypeEnum.kDefaultDisplayLengthUnits)

oParam = oUserParams.AddByValue("NewParam3", 3 * 2.54, _
    UnitsTypeEnum.kDefaultDisplayLengthUnits)
```

# Documentオブジェクトの概要

## パラメータ APIのみの機能

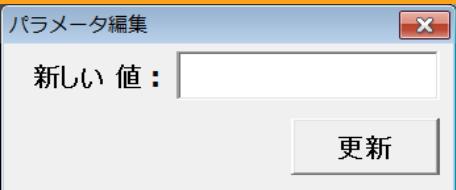
- モデルと参照パラメータの作成
- 未使用モデルと参照済みパラメータの削除
- DisabledActionTypes…ユーザーパラメータの削除禁止
- Dependents, DrivenBy…パラメータ間の依存関係情報を提供
- カスタムパラメータグループの作成
- パラメータタイプの変更
  - ✓ モデルから参照
  - ✓ 参照からモデル
  - ✓ ユーザーからモデル
  - ✓ ユーザーから参照



# 演習6

## ParameterとUOM

- パーツのサイズをコントロールする“Length”というパラメータを含む、新規パートファイルを作成してください
- 以下のようなダイアログを含むVBAプログラムを作成してください



- エンドユーザーが妥当な値を入力できるようにします。入力値が無効な場合、フィードバックとしてテキストを赤色で表示します
- “更新”ボタンが押されると、パラメータを更新します

- 演習のヒント
  - ✓ UOM.CompatibleUnitsメソッドで入力値の妥当性を確認可能
  - ✓ ドキュメントオブジェクトのComponentDefinition.Parameters.Item("Length")により、Lengthパラメータにアクセスが可能



パートドキュメント

# パートドキュメント

本章のゴール～APIで以下の形状を作成

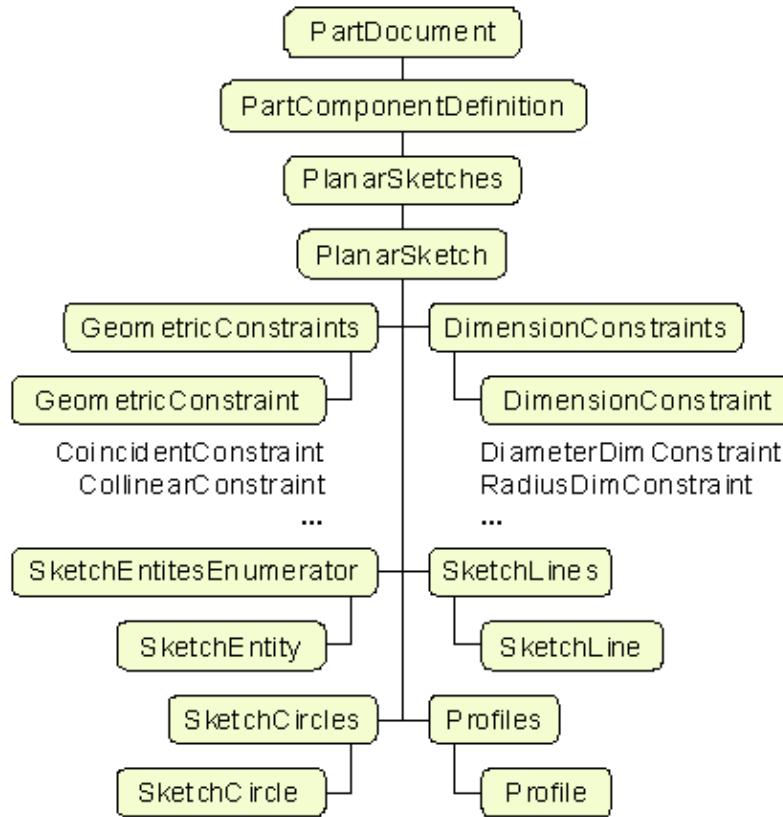


## 作成手順

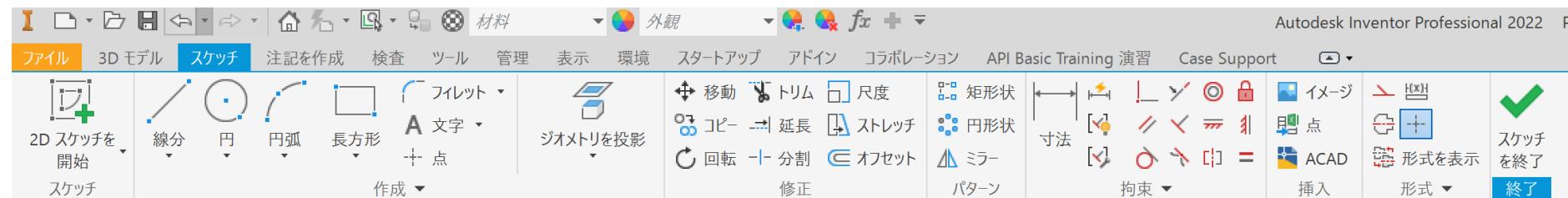
1. 押し出し1の形を定義するスケッチ1の作成(エンティティと拘束)
2. スケッチ1を使って押し出し1の作成
3. パーツのフェース上にスケッチ2を作成
4. スケッチ2を使って押し出し2の作成
5. エッジを選択しフィレット1の作成

# パートドキュメント

## スケッチ

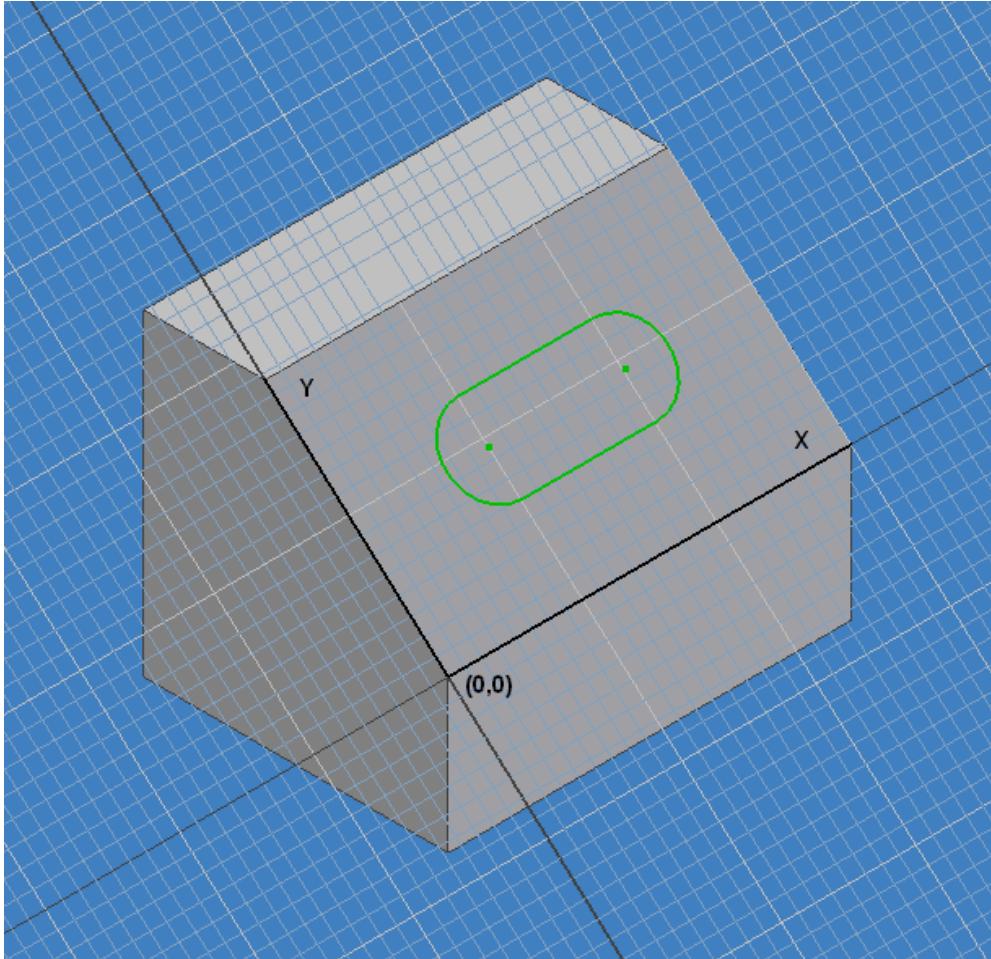


- スケッチは、大部分のフィーチャーの為のビルディングブロック
- スケッチは、ジオメトリ拘束と寸法拘束によって振る舞いが制御されるエンティティ(線、円など)により構成
- パーツドキュメントは、スケッチの基本クラスから派生したPlanarSketchオブジェクトを使用



# パートドキュメント

## スケッチ



- スケッチは2次元のジオメトリを持つ
- スケッチは自身の2次元座標系を定義
- スケッチはパートの平面ジオメトリに関する3次元D空間に置かれる(フェース面または作業平面)
- スケッチは、パートまたはアセンブリの3次元空間内の平面ジオメトリ(フェース面または作業平面)に位置する

# パーティドキュメント

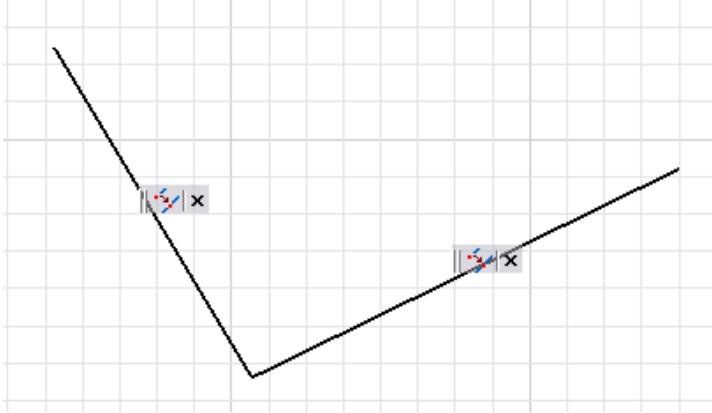
## スケッチの作成

- Add(PlanarEntity As Object, [UseFaceEdges As Boolean = False]) As PlanarSketch
  - ✓ 2D Sketch コマンドに相当
  - ✓ 方向は自動的に決められる
  - ✓ 入力フェースのエッジを任意に含むことが可能
- AddWithOrientation(PlanarEntity As Object, AxisEntity As Object \_  
,NaturalAxisDirection As Boolean, AxisIsX As Boolean, \_  
Origin As Object, [UseFaceEdges As Boolean = False]) As PlanarSketch
  - ✓ API のみの機能で、方向の制御に使用するエンティティを指定することで、方向を指定してのスケッチ作成が可能
  - ✓ **PlanarEntity**: スケッチを作成する平面オブジェクトを定義する Object 型の入力値です。この引数に対する有効な入力値には、平面と作業平面が含まれます
  - ✓ **AxisEntity**: スケッチ平面の X 軸または Y 軸を定義する Object 型の入力値です(どちらかを定義する AxisIsX 引数)。有効な入力値には、直線状エッジ、別のスケッチ線からのスケッチ線、作業軸が含まれます

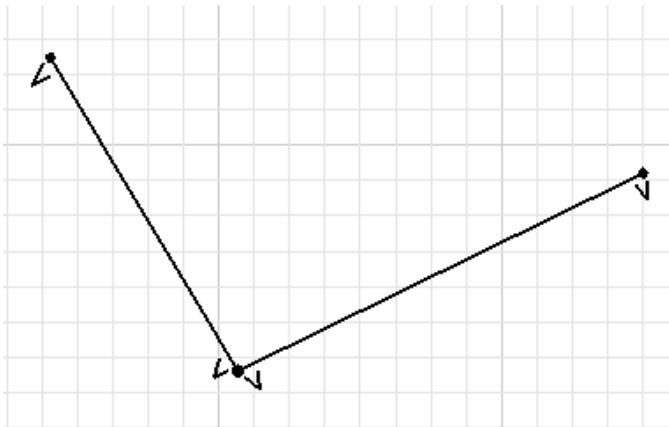
# パーティドキュメント

## スケッチエンティティの基礎

相互に影響する2つの線



3点と4つのジオメトリ拘束



- スケッチの構成要素
  - ✓ スケッチエンティティ
  - ✓ ジオメトリ拘束
  - ✓ 寸法拘束
- スケッチエンティティは、常にスケッチポイントと関連しており、かつ一致拘束点に接続されている

# パートドキュメント

## スケッチエンティティ

- スケッチエンティティの作成には、以下の2つの方法で座標系を定義することが可能
  1. Point2d オブジェクトを使用…Point2d オブジェクトで定義した座標系で、スケッチポイントを作成
  2. 既存の SketchPoint オブジェクトを使用…入力SketchPoint を使ってスケッチエンティティを作成

# パーティドキュメント

## モデル空間とスケッチ空間

- スケッチは、モデル空間内の2D 平面であり、以下のメソッドは、1つの空間から他へ移動する際に便利
  - ✓ ModelToSketchSpace…モデル空間内の3D ポイントから、スケッチ平面上に3D ポイントを投影した2Dポイントを返す
  - ✓ SketchToModelSpace…スケッチ平面内の2Dポイントから、モデル空間上の3D ポイントを返す
  - ✓ ModelToSketchTransform、SketchToModelTransform…これらの変換を定義した変換マトリクスを返す

# パートドキュメント

## スケッチのジオメトリ拘束

- 単一エンティティの振る舞いの定義：固定，垂直，水平

```
Dim oHorizontal As HorizontalConstraint  
oHorizontal = oSketch.GeometricConstraints.AddHorizontal(oSketchLine1)
```

- 2つのエンティティ間の関連定義：同一点，同一線上，同心円，長さが等しい，角度が等しい，水平の一列，中心点，平行，垂線，対称，接線，垂直の一列

```
Dim oParallel As ParallelConstraint  
oParallel = oSketch.GeometricConstraints.AddParallel(oSketchLine1, oSketchLine2)
```



# パートドキュメント



Dimension

## スケッチの寸法拘束

- エンティティのサイズと距離、またはエンティティ間の角度の定義
- 多くの寸法拘束はスケッチ点に設定
- プログラミングヘルプに、様々な寸法拘束の記述例を提供

```
Dim oTwoPointDim As TwoPointDistanceDimConstraint  
oTwoPointDim = oSketch.DimensionConstraints.AddTwoPointDistance(_  
    oSketchLine1.StartSketchPoint, oSketchLine1.EndSketchPoint, _  
    DimensionOrientationEnum.kAlignedDim, oTG.CreatePoint2d(3, 4), False)
```

- 寸法拘束
  - ✓ DiameterDimConstraint
  - ✓ EllipseRadiusDimConstraint
  - ✓ OffsetDimConstraint Inventor
  - ✓ OffsetSplineDimConstraint
  - ✓ RadiusDimConstraint
  - ✓ TangentDistanceDimConstraint
  - ✓ ThreePointAngleDimConstraint
  - ✓ TwoLineAngleDimConstraint
  - ✓ TwoPointDistanceDimConstraint

# パートドキュメント

## スケッチ作成サンプル SketchLine使用(VB.net)

```
Dim oCompDef As PartComponentDefinition
oCompDef = _InvApplication.ActiveDocument.ComponentDefinition

Dim oSketch As PlanarSketch
oSketch = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

Dim oTG As TransientGeometry
oTG = _InvApplication.TransientGeometry

Dim oLines(2) As SketchLine
oLines(0)= oSketch.SketchLines.AddByTwoPoints(oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(3, 0))
oLines(1)= oSketch.SketchLines.AddByTwoPoints(oLines(0).EndSketchPoint, oTG.CreatePoint2d(0, 2))
oLines(2)= oSketch.SketchLines.AddByTwoPoints(oLines(1).EndSketchPoint, oLines(0).StartSketchPoint)

Call oSketch.GeometricConstraints.AddHorizontal(oLines(0))
Call oSketch.GeometricConstraints.AddVertical(oLines(2))

Call oSketch.DimensionConstraints.AddTwoPointDistance(oLines(0).StartSketchPoint, _
           oLines(0).EndSketchPoint, _
           DimensionOrientationEnum.kHorizontalDim, _
           oTG.CreatePoint2d(1.5, -0.5))

Call oSketch.DimensionConstraints.AddTwoPointDistance(oLines(2).StartSketchPoint, _
           oLines(2).EndSketchPoint, _
           DimensionOrientationEnum.kVerticalDim, _
           oTG.CreatePoint2d(-0.5, 1))
```

# パートドキュメント

## スケッチ作成サンプル SketchPoint使用(VB.net)

```
Dim oPoints(2) As SketchPoint
oPoints(0) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 0), False)
oPoints(1) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(3, 0), False)
oPoints(2) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 2), False)

Dim oLines(2) As SketchLine
oLines(0) = oSketch.SketchLines.AddByTwoPoints(oPoints(0), oPoints(1))
oLines(1) = oSketch.SketchLines.AddByTwoPoints(oPoints(1), oPoints(2))
oLines(2) = oSketch.SketchLines.AddByTwoPoints(oPoints(2), oPoints(0))

Call oSketch.GeometricConstraints.AddHorizontal(oLines(0))
Call oSketch.GeometricConstraints.AddVertical(oLines(2))

Call oSketch.DimensionConstraints.AddTwoPointDistance(oPoints(0), _
    oPoints(1), _
    DimensionOrientationEnum.kHorizontalDim, _
    oTG.CreatePoint2d(1.5, -0.5))

Call oSketch.DimensionConstraints.AddTwoPointDistance(oPoints(2), _
    oPoints(0), _
    DimensionOrientationEnum.kVerticalDim, _
    oTG.CreatePoint2d(-0.5, 1))
```

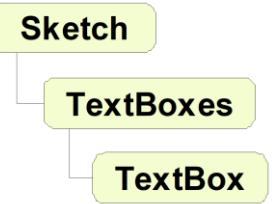
# パートドキュメント

## スケッチエンティティの作成と編集

- *DeferUpdates*プロパティでスケッチソルバの動作を止めることによってパフォーマンスを改善。ただし、スケッチソルバを必要とする編集に注意。スケッチの新規作成時に使用することを推奨
- 位置の編集は、拘束の編集または、拘束されていないスケッチポイントを移動することで実施
- スケッチエンティティセットの移動または回転は、*MoveSketchObjects*メソッドを使う事で可能。本メソッドで、点を移動または回転させることで信頼できる結果を得ることができる(添付されたジオメトリは自動的に更新される)
- スケッチエンティティの*ConstraintStatus*プロパティを使うことで、エンティティが完全に拘束されたを判定することが可能
- *CopyContentsTo*により、スケッチのすべてのコンテンツをコピーが可能

# パーティドキュメント

## スケッチ内のText



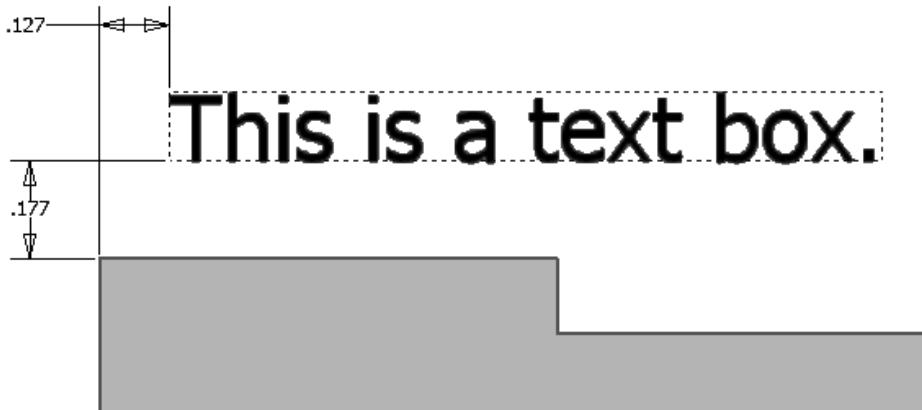
- TextBoxの文字列は、XMLシンタックスを使うことによって、文字内の定義スタイルの変更が可能。詳細は、オンラインヘルプを参照
- サンプルコード(VB.net)

```
' シンプルな文字列のテキストを作成
' テキストのオーバライドを使用せず、アクティブなテキストスタイルを使用
Dim sText As String
sText = "Drawing Notes"
Dim oTextBox As TextBox
oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 18), sText)

' オーバライドを使ってテキストを作成
sText = "<StyleOverride Bold='True' Italic='True'>Notice</StyleOverride>: " & _
"Some Text" & _
"<StyleOverride Font='AIGDT'>n</StyleOverride>" & _
"Some other Text"
oTextBox = oSketch.TextBoxes.AddFitted(oTG.CreatePoint2d(3, 16), sText)
```

# パーティドキュメント

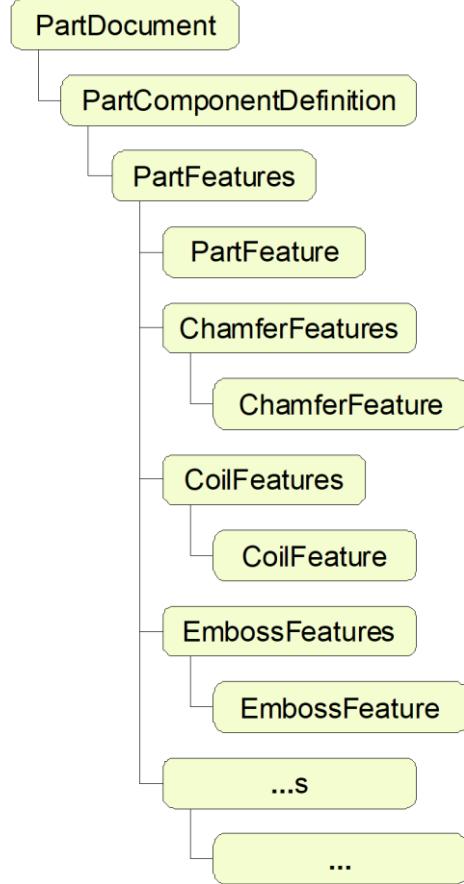
## テキストボックス



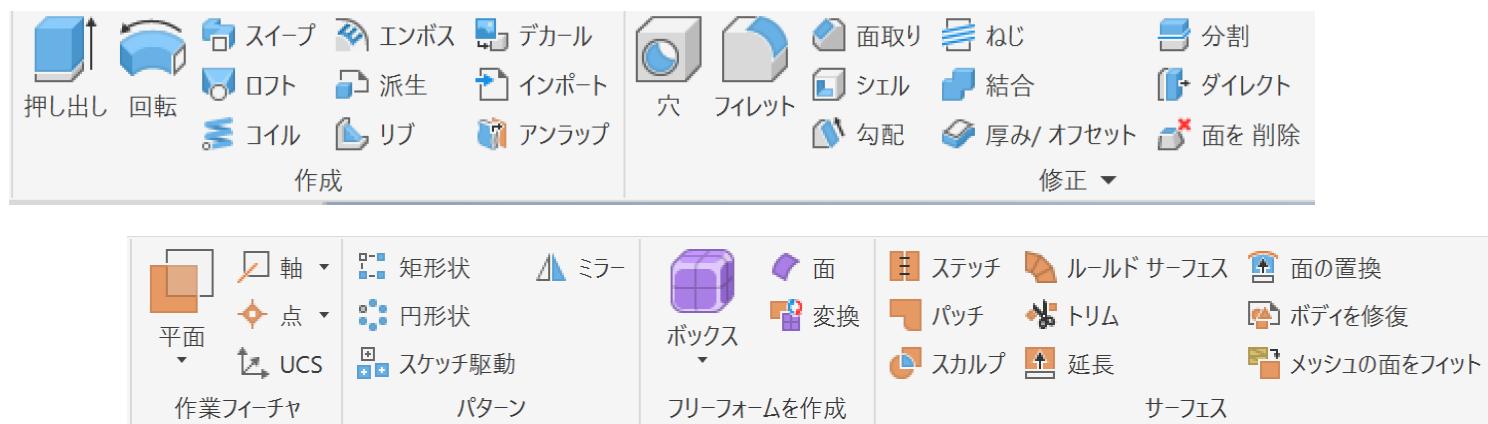
- *ShowBoundaries*プロパティを使用し、テキストボックスの境界の周りの可視性をトグルで切り替える(スケッチ線)
- *BoundaryGeometry*プロパティを使用し、境界ボックスのスケッチラインにアクセス

# パートドキュメント

## フィーチャー

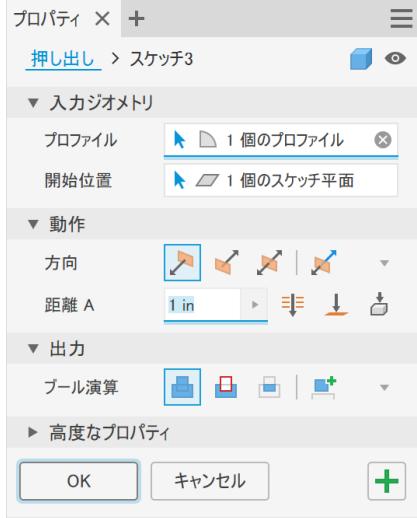


- ほとんどのフィーチャーで作成と高度な問い合わせが利用可能
- ほとんどのフィーチャーで、一般の問い合わせ機能が利用可能
- フィーチャーは、ブラウザーに現れた同じ順序で *PartFeatures*コレクションを通して列挙されている

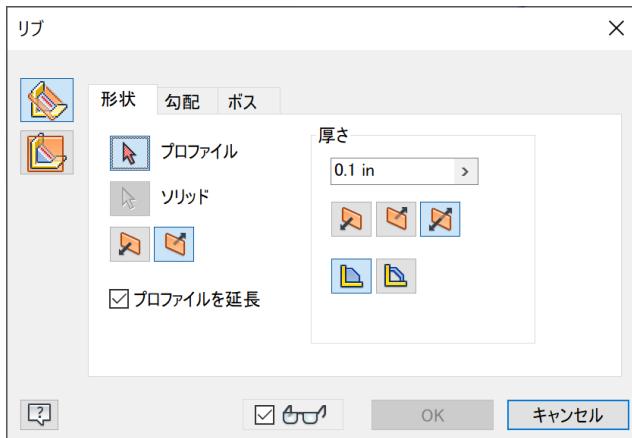


# パートドキュメント

## Sketchベースのフィーチャ

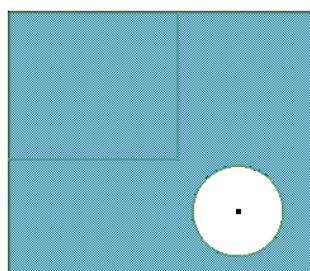
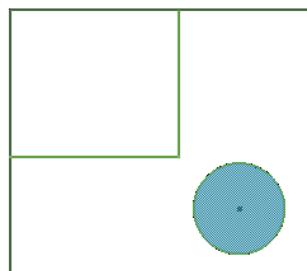
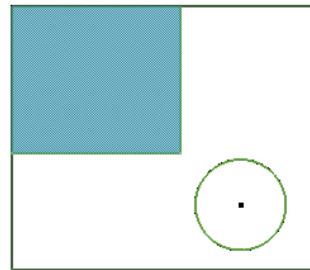
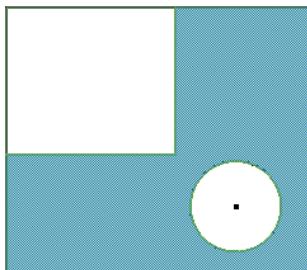
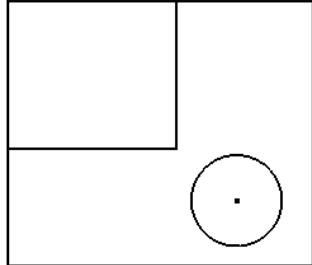


- 多くのフィーチャーは、入力として形状またはフィーチャーの位置を定義するスケッチが必要
  - ✓ Extrude, Revolve, Hole, Rib, Loft, Sweep, Coil, Split, Emboss, Boundary
- APIでのフィーチャーの作成は、ユーザーインターフェースを使ってフィーチャーを作成する場合と同じ入力が必要



# パートドキュメント

## プロファイル

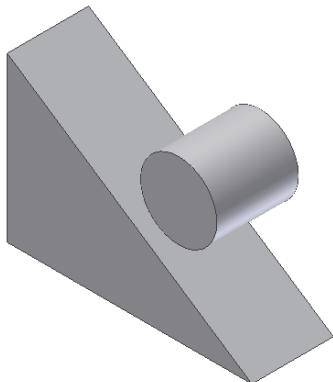


- フィーチャーの作成は、スケッチではなくプロファイルとパスを入力とします
- プロファイルはスケッチ内の開いたまたは閉じたループを定義して、フィーチャーが使用
- 閉じたプロファイルは、閉じたループの組み合わせで構成
- 開いたプロファイルは、1つの終端が繋がったエンティティのセットから構成

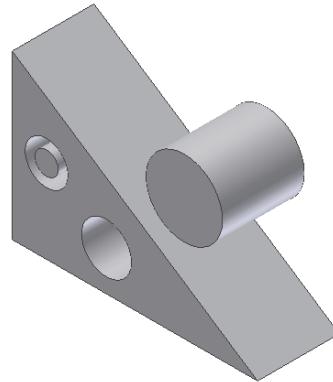
# パートドキュメント

## プロファイルの作成

Combine = False



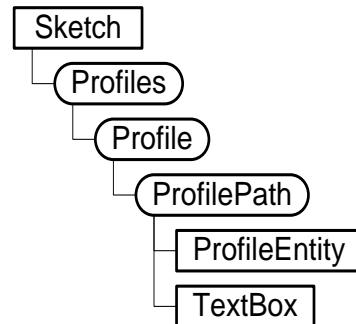
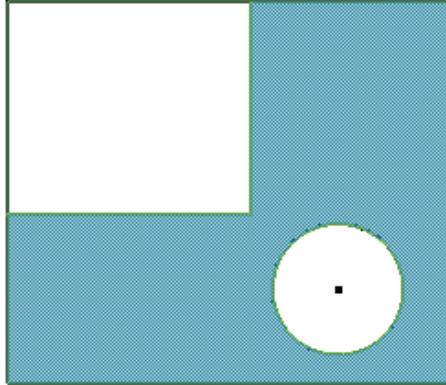
Combine = True



- *Profiles.AddForSolid([Combine As Boolean = True], [Curves], [reserved]) As Profile*
  - 閉じたスケッチエンティティセットからプロファイルを作成。テキストボックスを含んでいるスケッチをサポート
  - Curves 引数を使用して、プロファイル内に含んで欲しいスケッチエンティティを明示する
  
- *Profiles.AddForSurface ([Curve]) As Profile*
  - 連続したスケッチエンティティのオープンセットからプロファイルの作成

# パーティドキュメント

## プロファイルのオブジェクトモデル



- 既存のプロファイルは、1つ以上の*ProfilePath*オブジェクトで構成され、*ProfilePath*オブジェクトは、プロファイルの（開いたまたは閉じた）ループ表す
- ProfilePath*オブジェクトは、プロファイルからパスを取り去るために削除が可能。*AddsMaterial*プロパティは修正する事も可能
- ProfilePath*は、1つ以上の*ProfileEntity*オブジェクトまたはテキストボックスで構成
- PlanarSketch.UpdateProfiles*は、全ての関連したプロファイルや依存しているフィーチャーを再計算する (APIを利用しスケッチ編集後に使用)
- プロファイルは、Drawingで塗りつぶしのエリアを定義する用途でも使用される

# パートドキュメント

## フィーチャーの作成 **Definition**を用いた方法

- 押し出しのような、幾つかのフィーチャーは “Definition” コンセプトに基づく
- 新規設定/プロパティのサポートを提供し、将来的なフィーチャー機能の拡張に容易に対応する API
- 手順
  1. Definitionの作成 例) `ExtrudeFeatures.CreateExtrudeDefinition`
  2. 押し出し定義の属性・様々なオプションを設定
  3. Definitionをもとにフィーチャーを追加 例) `ExtrudeFeatures.Add .`
- 全機能に対するフルサポートを提供

# パートドキュメント

## 押し出しサンプル (VB.net)

- 指定した距離で押し出しを押し出しを作成

```
'get ExtrudeFeatures collection
Dim extrudes As ExtrudeFeatures
extrudes = oPartDocDef.Features.ExtrudeFeatures

' Create an extrude definition.
Dim extrudeDef As ExtrudeDefinition
extrudeDef = extrudes.CreateExtrudeDefinition(rectProfile, kNewBodyOperation)

' Modify the extent and taper angles.
extrudeDef.SetDistanceExtent(8, kNegativeExtentDirection)
extrudeDef.SetDistanceExtentTwo(20)
extrudeDef.TaperAngle = "-2 deg"
extrudeDef.TaperAngleTwo = "-10 deg"

' Create the extrusion.
Dim extrude As ExtrudeFeature
extrude = extrudes.Add(extrudeDef)
```

# パートドキュメント

## フィーチャーの作成 直接作成する方法

- 幾つかのフィーチャは、直接作成する方法を使用
- フィーチャを作成するAPIは、フィーチャーをUIで作成する場合と同様の入力を必要とする

例) *RevolveFeatures.AddByAngle* メソッド

*AddByAngle(Profile As Profile,*

*AxisEntity As IDispatch,*

*Angle As VARIANT,*

*ExtentDirection As PartFeatureExtentDirectionEnum,*

*Operation As PartFeatureOperationEnum,*

*Result As [out, retval] RevolveFeature\*)*

# パートドキュメント

## 回転フィーチャー( Revolve Feature)の作成 コードサンプル(VB.net)

- 角度を指定して、回転フィーチャーを作成

```
'get RevolveFeatures collection
Dim revolves As RevolveFeatures
revolves = oPartDocDef.Features.RevolveFeatures

' Create the Revolve Feature
Dim revolve As RevolveFeature
revolve = revolves.AddByAngle(oProfile,
                             oAxisEntity,
                             360,
                             PartFeatureExtentDirectionEnum.kNegativeExtentDirection,
                             PartFeatureOperationEnum.kJoinOperation)
```

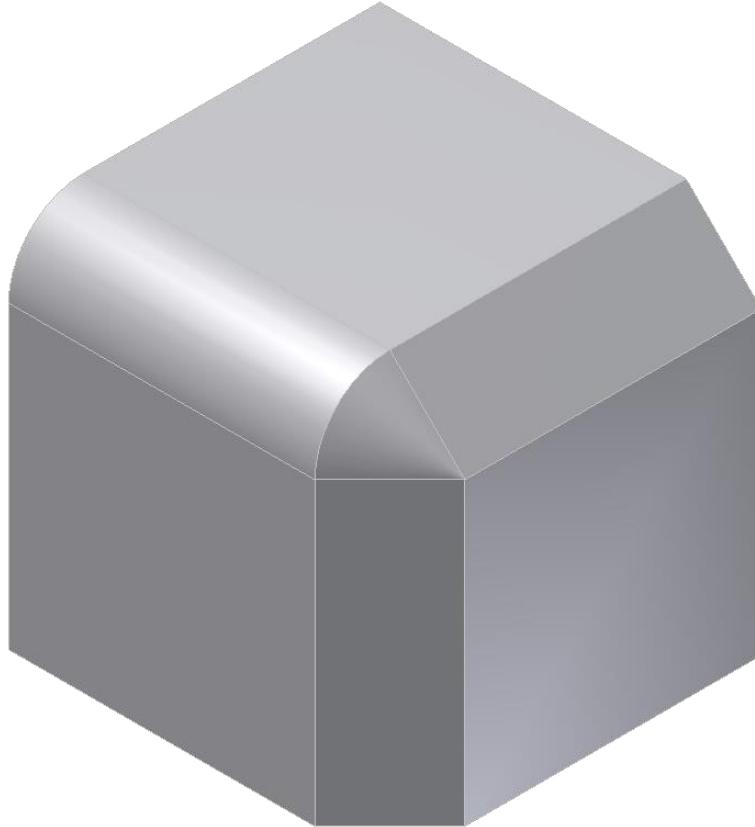
# パートドキュメント

## フィーチャー編集

- フィーチャーの編集には幾つかの方法がある
  - ✓ 関連したパラメータの値を編集
  - ✓ 元となるスケッチの内容の編集(変更の内容によっては、`PlanarSketch.UpdateProfiles` を呼び出す必要がある)
  - ✓ 範囲タイプの変更(押し出しと回転フィーチャーでのみサポート)
  - ✓ フィーチャを挿入するために、フィーチャツリー内の停止ノード再配置  
*PartFeature.SetEndOfPart*
  - ✓ フィーチャーの削除

# パートドキュメント

## Local Modification フィーチャ



- Local modificationフィーチャーは、プロファイルまたはパス入力を必要としない既存モデルの修正を実行
  - ✓ 例: Shell, Thread, Fillet, その他
- 入力引数として、既存のジオメトリが必要

# パートドキュメント

## フィレットの作成 サンプル(VB.Net)

```
Dim oPartDoc As PartDocument
oPartDoc = _InvApplication.ActiveDocument

Dim oPartDef As PartComponentDefinition
oPartDef = oPartDoc.ComponentDefinition

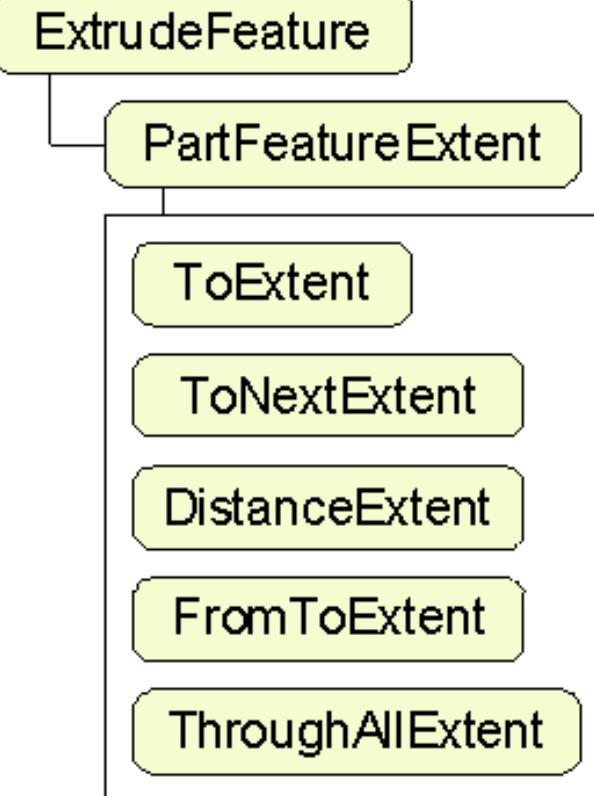
'PickFirst でエッジが選ばれると想定
Dim oEdge As Edge
oEdge = oPartDoc.SelectSet.Item(1)

'エッジコレクションの作成
Dim oEdges As EdgeCollection
oEdges = ThisApplication.TransientObjects.CreateEdgeCollection
oEdges.Add(oEdge)

Dim oFillet As FilletFeature
oFillet = oPartDef.Features.FilletFeatures.AddSimple(oEdges, 0.5)
```

# パートドキュメント

## Definition/Extent(定義 範囲) オブジェクト

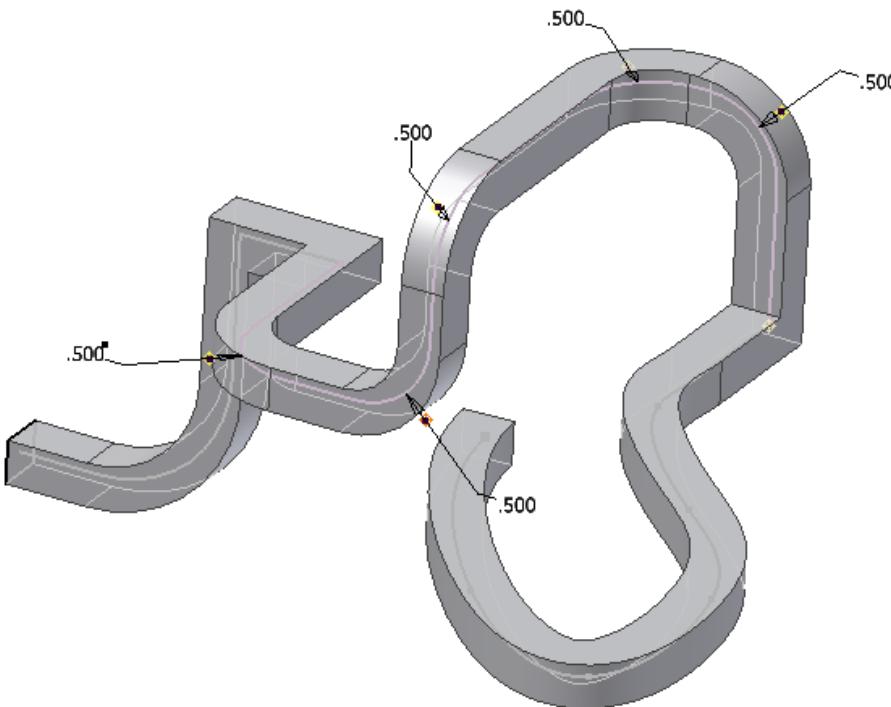


- いくつかのオブジェクトは、様々な定義方法のうちのひとつでの定義され、その定義は場合によって変えることが可能
- 例として、押し出しフィーチャーの範囲は、Distance, To Next, From To , All として定義できる。同じ押し出しフィーチャーであるが、範囲タイプは様々
- その他のフィーチャー( revolve, coil, hole, extrude)も、それぞれ異なる範囲タイプを持っている
- その他オブジェクト( iFeatures, derived parts, work features, holes, chamfer, fillet, shell)もそれらの定義に、同様の概念を使う

# パートドキュメント

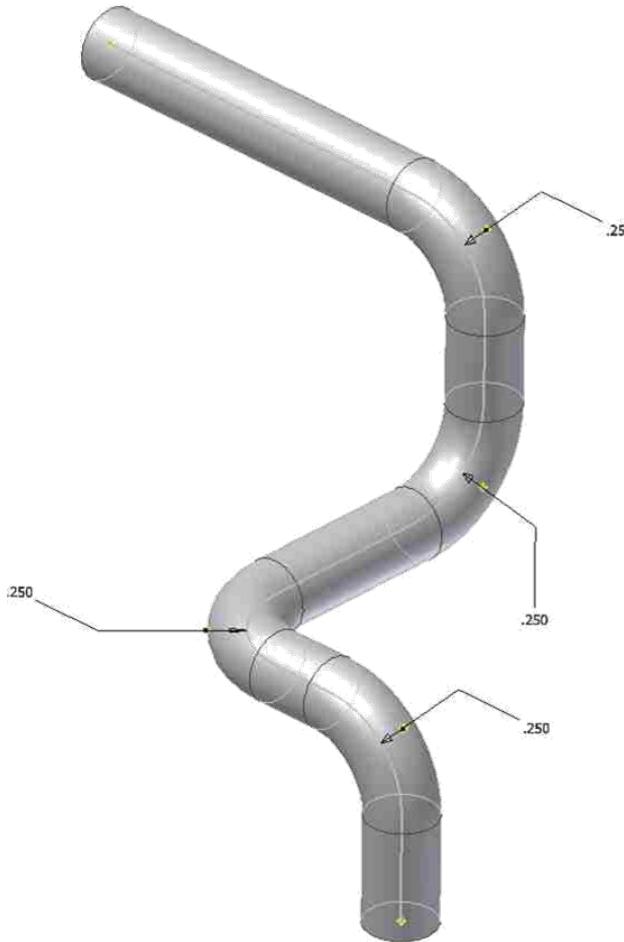
## パス(Path)

- パスは、位置的に繋がれた2Dおよび3Dスケッチの組み合わせから構成
- スイープ、分割フィーチャー、矩形状パターンフィーチャーの入力に使用される
- Inventorは作成時に、1つのエンティティを入力として、エンティティの全て終端を繋いだ鎖を見つける



# パートドキュメント

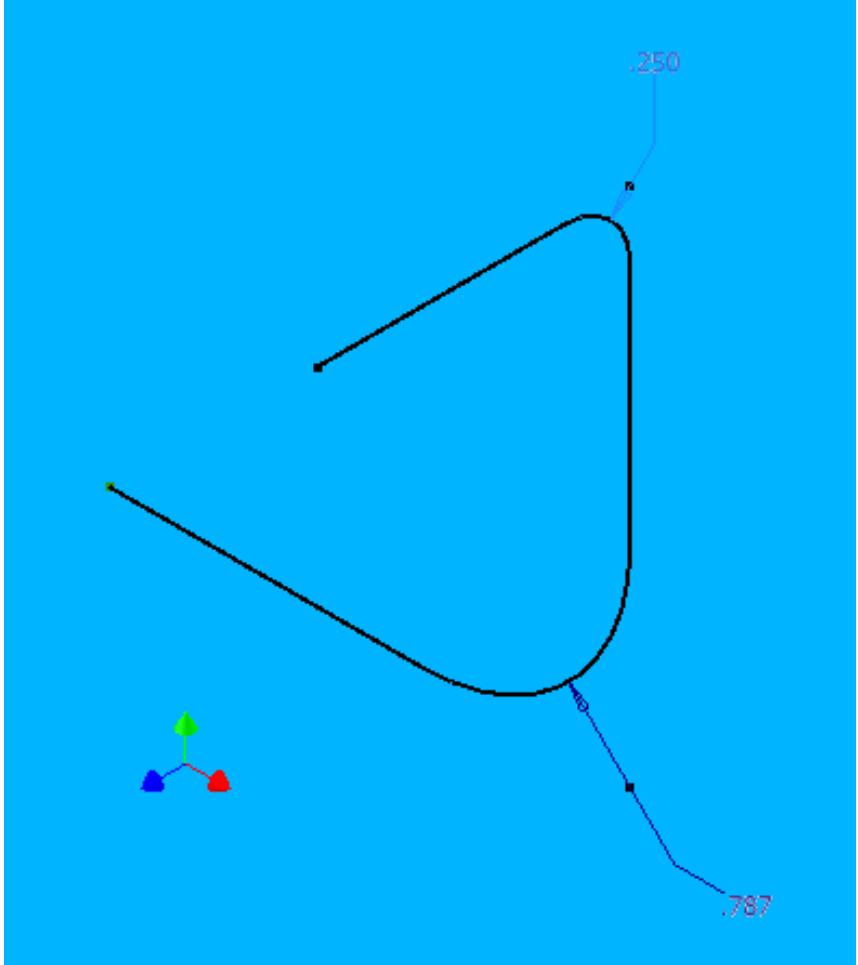
## 3Dスケッチ



- APIは3Dスケッチの作成、検索、編集をサポート
- 2Dスケッチと同様に、3Dスケッチは常に点群(workpoints, 2D スケッチポイントまたは、モデル頂点)に接続されている
- 線分、スプライン、固定スプラインから作成可能
- モデルまたは、他のスケッチの既存のジオメトリを含むことが可能
- ジオメトリ拘束、寸法拘束を作成可能

# パートドキュメント

## 3Dスケッチ サンプルコード(VB.Net)



```
Public Sub Draw3DSketch()
    'active documentの取得
    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.ActiveDocument
    'part component definitionの取得
    Dim oPartDef As PartComponentDefinition
    oPartDef = oPartDoc.ComponentDefinition
    'sketch 3d追加
    Dim oSketch3D As Sketch3D
    oSketch3D = oPartDef.Sketches3D.Add
    Dim oTG As TransientGeometry
    oTG = _InvApplication.TransientGeometry
    '3D sketch line追加
    Dim oLastLine As SketchLine3D
    oLastLine = oSketch3D.SketchLines3D.AddByTwoPoints( _
        oPartDef.WorkPoints.Item(1), _
        oTG.CreatePoint(5, 0, 0))
    ' 最初のsketck lineの端点を使用して3D sketch lineを追加
    oLastLine = oSketch3D.SketchLines3D.AddByTwoPoints( _
        oLastLine.EndSketchPoint, _
        oTG.CreatePoint(5, 5, 0), _
        True, 2)
    '2番目のsketck lineの端点を使用して3D sketch lineを追加
    oLastLine = oSketch3D.SketchLines3D.AddByTwoPoints( _
        oLastLine.EndSketchPoint, _
        oTG.CreatePoint(5, 5, 3))
End Sub
```

# パートドキュメント

## スイープフィーチャー(Sweep Features)

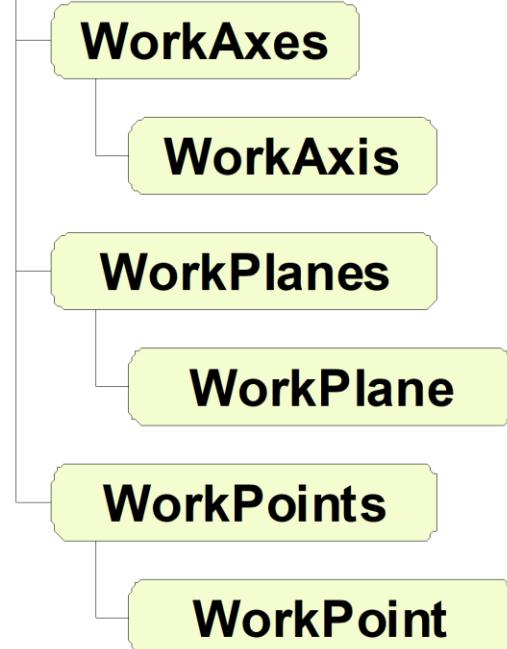
- SweepFeatures コレクションは様々なスイープを作成する方法を提供
  - ✓ *AddUsingPath (...)*
  - ✓ *AddUsingPathAndGuideRail (...)*
  - ✓ *AddUsingPathAndGuideSurface (...)*
  - ✓ *AddUsingPathAndSectionTwists (...)*
- 様々なユーティリティメソッドも提供
  - ✓ *GetProfileOrientationAtParam (...)*
  - ✓ *GetTruePath (...)*

# パートドキュメント

## ワークフィーチャー(Work Features)



### PartComponentDefinition



- ワークフィーチャーの全てのタイプで、完全なオブジェクトの問い合わせと編集がサポートされる:planes, axes, points
- 基礎となる作業点に加えて、API は基礎になる作業平面と作業軸をサポート
- アセンブリドキュメントでは、唯一AddFixed メソッドをサポート。これは、アセンブリ内に作業平面を作成し、アセンブリ拘束を位置のコントロールに使用

# パートドキュメント

## 既存のワークフィーチャへのアクセス

- ワークフィーチャーは、ブラウザの表示と同じ順序のコレクションによって列挙される
- 名前によってもアクセス可能(名前は、Inventorの言語によって異なる)

```
Public Sub GetWorkFeatures()

    Dim oPartDef As PartComponentDefinition
    oPartDef = _InvApplication.ActiveDocument.ComponentDefinition

    ' X-Y 平面の作業平面を得る
    Dim oXYPlane As WorkPlane
    oXYPlane = oPartDef.WorkPlanes.Item(3)
    Debug.Print(oXYPlane.Name)

    ' 原点を取得
    Dim oOriginPoint As WorkPoint
    oOriginPoint = oPartDef.WorkPoints.Item(1)
    Debug.Print(oOriginPoint.Name)

End Sub
```

# パートドキュメント

## ワークフィーチャの作成

- 様々なAdd メソッドにより、ワークフィーチャーの異なる作成方法をサポート
- Construction引数に“True”を指定することで、見えないワークフィーチャーを作成

```
Public Sub CreateWorkFeatures()

    Dim oPartDef As PartComponentDefinition
    oPartDef = _InvApplication.ActiveDocument.ComponentDefinition

    Dim oPlane1 As WorkPlane
    oPlane1 = oPartDef.WorkPlanes.AddByPlaneAndOffset(oPartDef.WorkPlanes.Item(1), 4)

    Dim oAxis1 As WorkAxis
    oAxis1 = oPartDef.WorkAxes.AddByTwoPlanes(oPlane1, oPartDef.WorkPlanes.Item(2))

    Dim oPlane2 As WorkPlane
    oPlane2 = oPartDef.WorkPlanes.AddByLinePlaneAndAngle(oAxis1, oPlane1, _
        System.Math.PI / 4)

End Sub
```

# パートドキュメント

## ワークフィーチャの参照/編集

### WorkPoints

#### WorkPoint

CurveAndEntityWorkPointDef

FixedWorkPointDef

MidPointWorkPointDef

NonLinearEdgeWorkPointDef

PointWorkPointDef

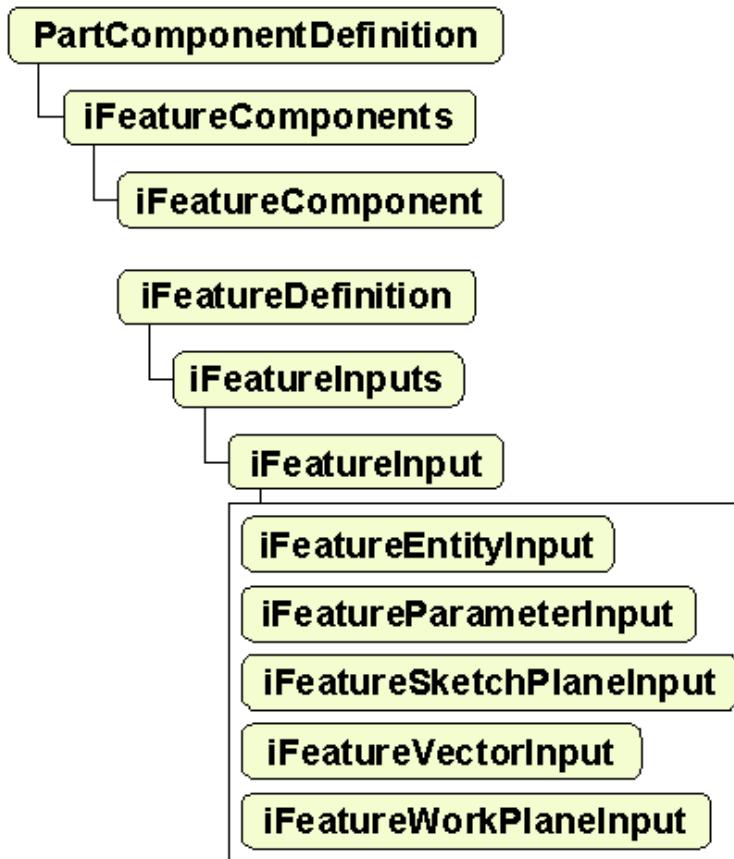
ThreePlanesWorkPointDef

TwoLinesWorkPointDef

- ワークフィーチャーオブジェクトは、全ワークフィーチャーに適用される一般的な機能をサポート
- どのようにワークフィーチャーが構成されているかや、参照エンティティへのアクセスを提供するDefinitionオブジェクトを、それぞれのワークフィーチャーから取得が可能
- ワークフィーチャーを編集するために、Definition オブジェクトのプロパティを編集することが可能
- 1つのタイプから他のタイプへワークフィーチャーを変更する様々なSetメソッドを利用可能

# パートドキュメント

## iFeature



- .ide ファイルによって *iFeatureDefinition* オブジェクトの作成
- 様々な入力セット
- *iFeatureDefinition*を使って *iFeatureComponent*を作成

# パートドキュメント

## iFeatureの作成サンプル(VB.Net)

```
Public Sub PlaceiFeature()
    oCompDef As PartComponentDefinition
    oCompDef= ThisApplication.ActiveDocument.ComponentDefinition

    oTG As TransientGeometry
    oTG= ThisApplication.TransientGeometry

    '最初の押し出しのスタートフェースを任意に得る。
    oFace As Face
    oFace= oCompDef.Features.ExtrudeFeatures.Item(1).StartFaces.Item(1)

    Dim oiFeatComps As iFeatureComponents
    oiFeatComps= oCompDef.ReferenceComponents.iFeatureComponents

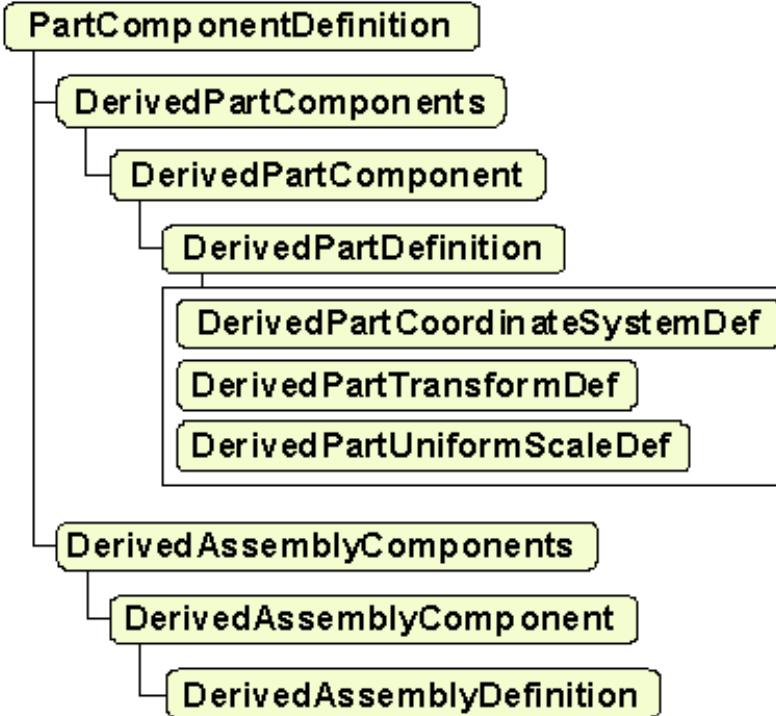
    '指定されたideファイルで定義オブジェクトの作成。
    Dim oiFeatDef As iFeatureDefinition
    oiFeatDef= oiFeatComps.CreateDefinition(_
        "C:\Users\Public\Documents\Autodesk\Inventor 2022\Catalog\Geometric
        Shapes\Cone_Rounded.ide")
```

```
'iFeature 入力値のセット。
Dim oiFeatInput As iFeatureInput
For Each oiFeatInput In oiFeatDef.iFeatureInputs
    Select Case oiFeatInput.Prompt
        Case "Pick a planar face or work plane"
            Dim oPlaneInput As iFeatureSketchPlaneInput
            oPlaneInput = oiFeatInput
            oPlaneInput.PlaneInput = oFace
            oPlaneInput.SetPosition(oTG.CreatePoint(0, 0, 0), _
                oTG.CreateVector(1, 0, 0), 0)
        Case "Enter Diameter"
            Dim oParamInput As iFeatureParameterInput
            oParamInput = oiFeatInput
            oParamInput.Expression = ".5 in"
        Case "Height at theoretical sharp point"
            oParamInput = oiFeatInput
            oParamInput.Expression = ".6 in"
        Case "Enter Radius"
            oParamInput = oiFeatInput
            oParamInput.Expression = ".1 in"
    End Select
    Next

    'iFeatureの作成。
    oiFeatComps.Add(oiFeatDef)
End Sub
```

# パートドキュメント

## パーティ・アセンブリの派生



- 派生パーティまたはアセンブリタイプを作成
- 様々な入力をセット
- *Definition*を使用し派生コンポーネントを作成

# パートドキュメント

## パート・センブリの派生 サンプル(VB.Net)

```
Public Sub InsertDerivedPart ()  
    Dim oCompDef As PartComponentDefinition  
    oCompDef = ThisApplication.ActiveDocument.ComponentDefinition  
    '派生パートの定義  
    Dim oDerivedPartComps As DerivedPartComponents  
    oDerivedPartComps = oCompDef.ReferenceComponents.DerivedPartComponents  
    '派生パートの読み込み  
    Dim oDerivedPartDef As DerivedPartUniformScaleDef  
    oDerivedPartDef = oDerivedPartComps.CreateUniformScaleDef ("C:\Temp\iPart.ipt")  
    'スケールセット  
    oDerivedPartDef.ScaleFactor = 0.75  
    '派生パートの追加  
    Dim oDerivedComp As DerivedPartComponent  
    oDerivedComp = oDerivedPartComps.Add(oDerivedPartDef)  
End Sub
```

# パートドキュメント

## iParts

- iPart factory テーブルへのアクセスの問い合わせ
- スプレッドシートにアクセスする事で、編集のためにExcelAPI を使用する事が可能
- iPartメンバーの作成
- アセンブリ内のiPart メンバーの配置

# パートドキュメント

## iPart インスタンスの作成と編集 (VB.Net)

```
Public Sub ChangeiPartMembers()
    'アセンブリドキュメントを得る。アセンブリがアクティブでなければ失敗。
    Dim oAsmDoc As AssemblyDocument
    oAsmDoc = ThisApplication.ActiveDocument

    'メンバーネームを使って、iPartメンバーを配置。
    Dim oOcc As ComponentOccurrence
    oOcc = oAsmDoc.ComponentDefinition.Occurrences.AddiPartMember( _
        " C:\Inventor API Training\Drawings\StandardFactory.ipt", _
        ThisApplication.TransientGeometry.CreateMatrix, "StandardFactory-02")
    MsgBox("iPartFactory-02配置")

    'インスタンスを、メンバーネームを使って別のメンバーに変更。
    oOcc.ChangeRowOfiPartMember("StandardFactory-03")
    MsgBox("iPartFactory-03に変更")

    'インスタンスを、テーブルのインデックスを使って別のメンバーに変更。
    oOcc.ChangeRowOfiPartMember(1)
    MsgBox("iPartFactory-01に変更")
End Sub
```

# パートドキュメント

## iPart メンバーのファイル作成 (VB.Net)

```
Public Sub CreateiPartMembers()
    Dim oDoc As PartDocument
    oDoc = ThisApplication.ActiveDocument
    'パートが iPart factory かを確認チェック
    If oDoc.ComponentDefinition.IsIPartFactory Then
        Dim oiPartFactory As iPartFactory
        oiPartFactory = oDoc.ComponentDefinition.iPartFactory
        'テーブル内の全てのメンバー列を通してイテレート
        Dim i As Long
        For i = 1 To oiPartFactory.TableRows.Count
            Dim oiPartMember As iPartMember
            '開いている iPart ファイルが存在する直下にホルダーを作成し
            'iPartメンバーファイルを作成
            oiPartMember = oiPartFactory.CreateMember(i)
        Next
    End If
End Sub
```

# 演習7

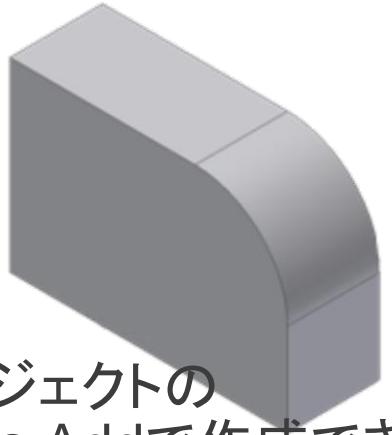
## スケッチとフィーチャーの作成

APIを用いて以下を実現してください

- 新規パーティドキュメントの作成
- X-Y平面上に、スケッチを作成
- パーツのメイン形状を定義するエンティティを作図(寸法は重要ではありません、拘束は任意に作成してください)
- 押し出しの作成
- 作成したファイルは、次の演習で使用します

### ■ 演習のヒント

- ✓ 新規スケッチはドキュメントオブジェクトの ComponentDefinition.Sketches.Add で作成できます。
- ✓ 押し出しがドキュメントオブジェクトの ComponentDefinition.Features.ExtrudeFeatures.AddByDistanceExtent で作成できます。



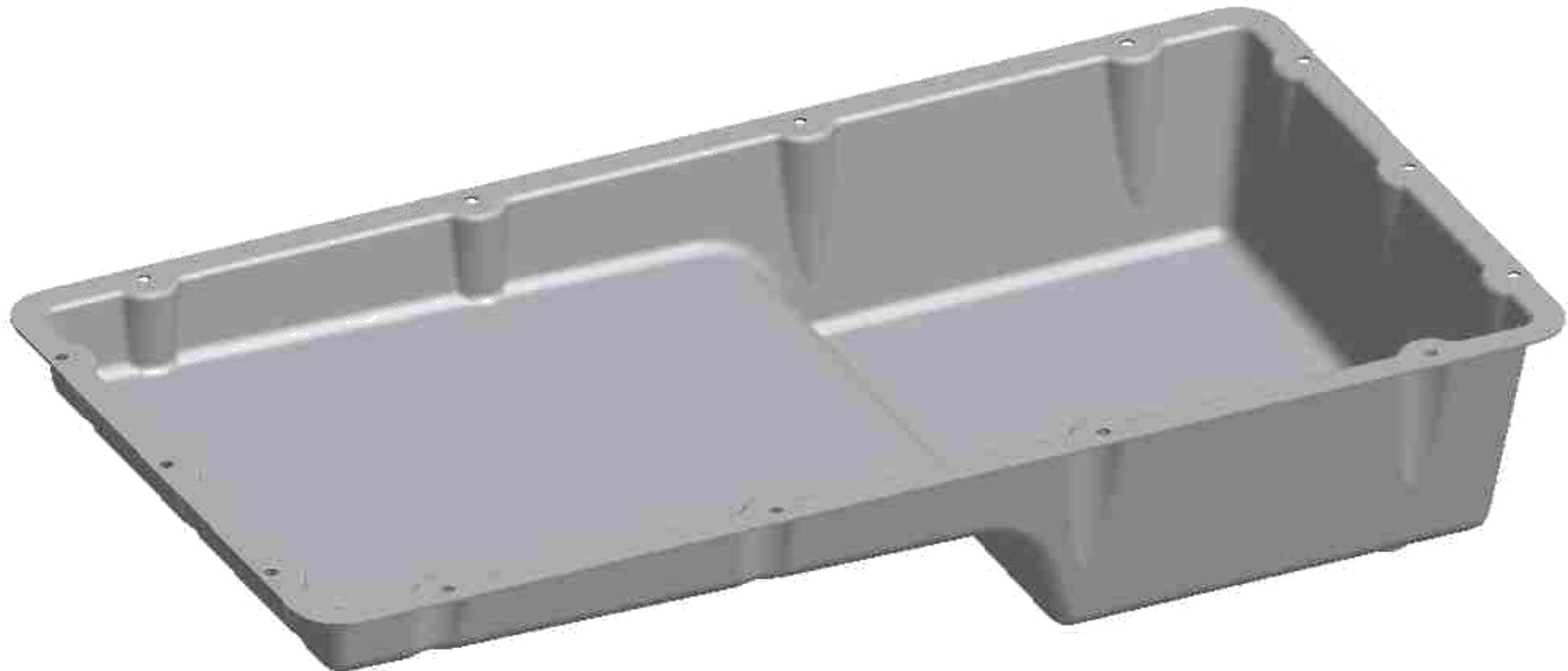


B-Rep

# B-Rep

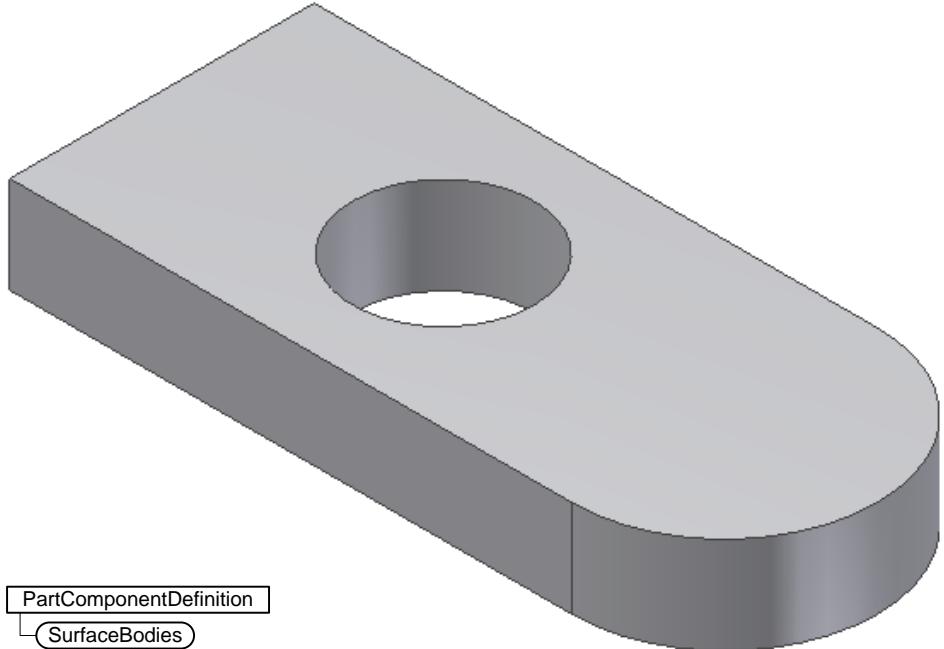
## B-Rep概要

- B-Rep(Boundary Representation)はソリッドモデルの完全なジオメトリの記述を提供
- ソリッドモデルのトポロジおよびジオメトリの両方へのアクセスを提供



# B-Rep

## SurfaceBody

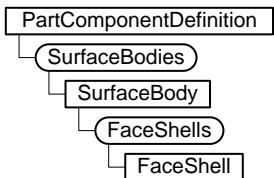
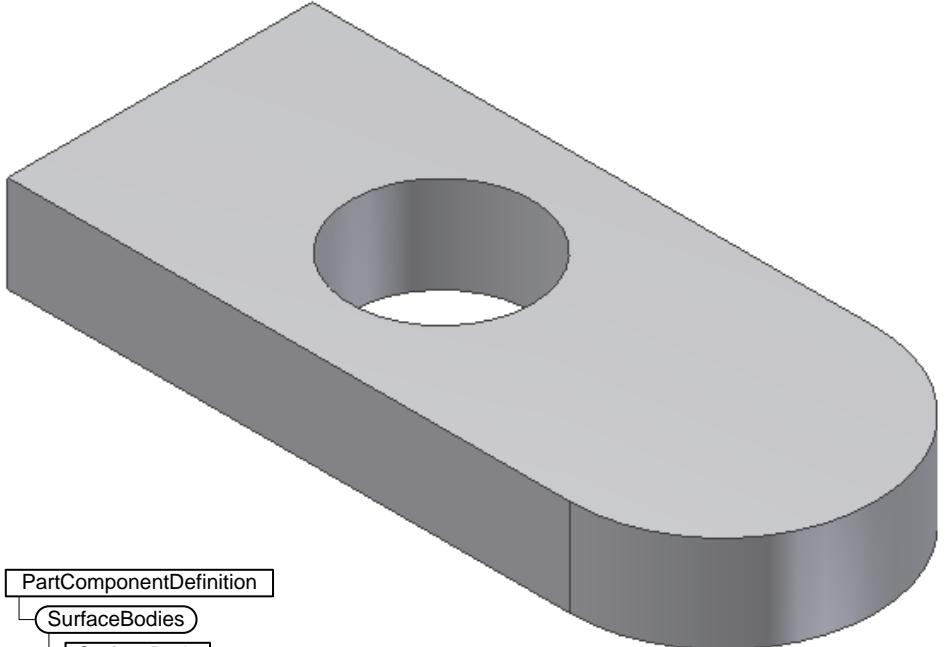


PartComponentDefinition  
└ SurfaceBodies  
  └ SurfaceBody

- 階層の最上位オブジェクトを定義
- ソリッド全体を表現
- ComponentDefinition オブジェクトから取得

# B-Rep

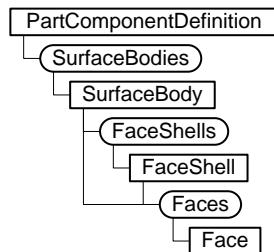
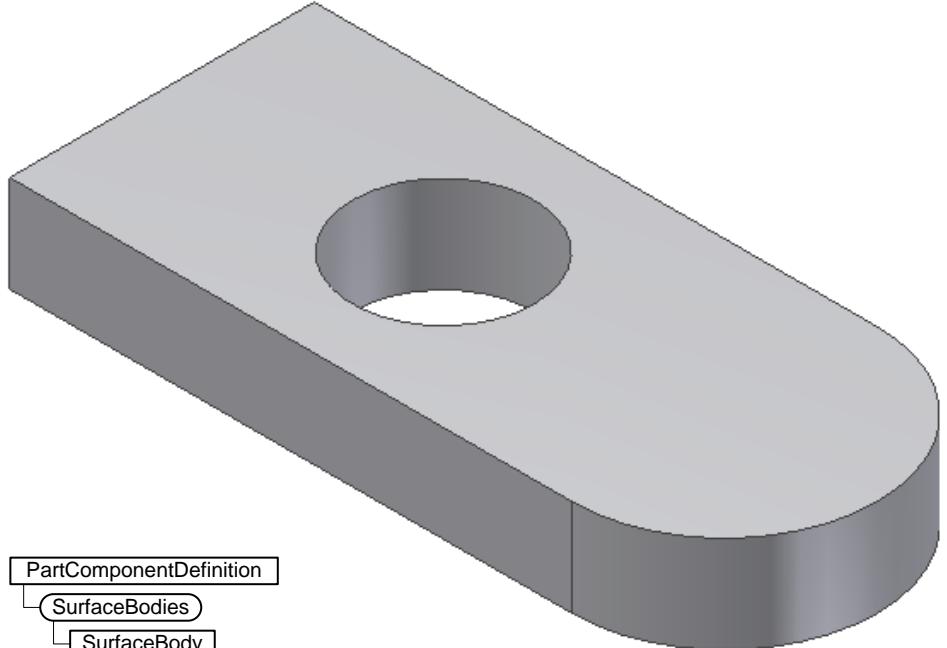
## FaceShell



- 接続されたFace(面)のセットを定義
- ほとんどのソリッドは1つのFaceShellを持つ

# B-Rep

## Face

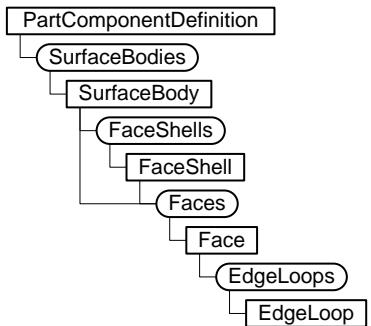
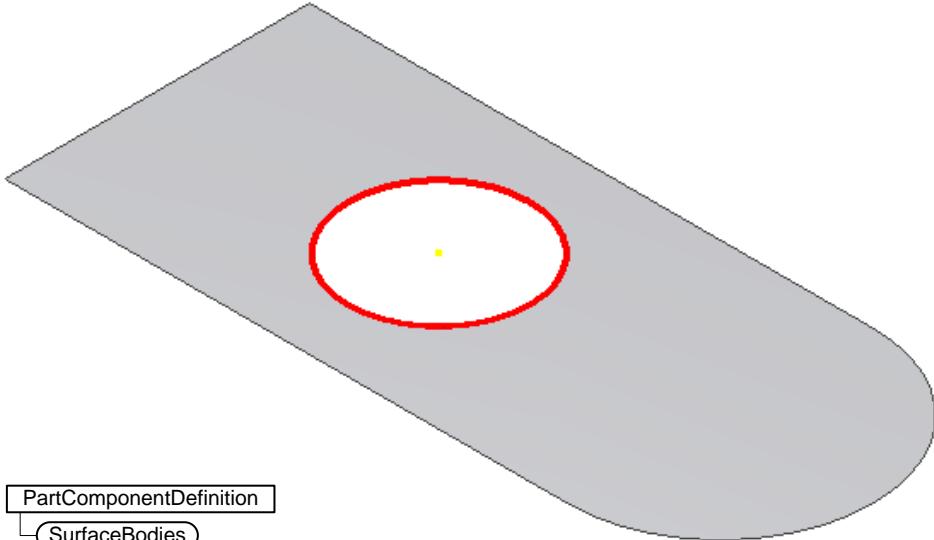


- 接続されたFace(面)がFaceShell を定義
- 容積を含む

# B-Rep

## EdgeLoop

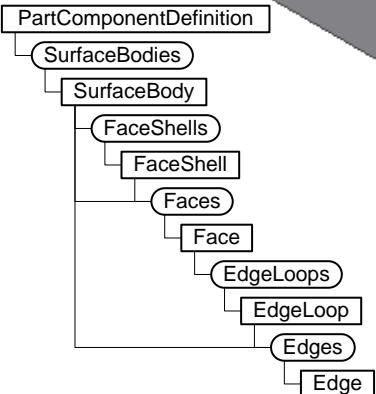
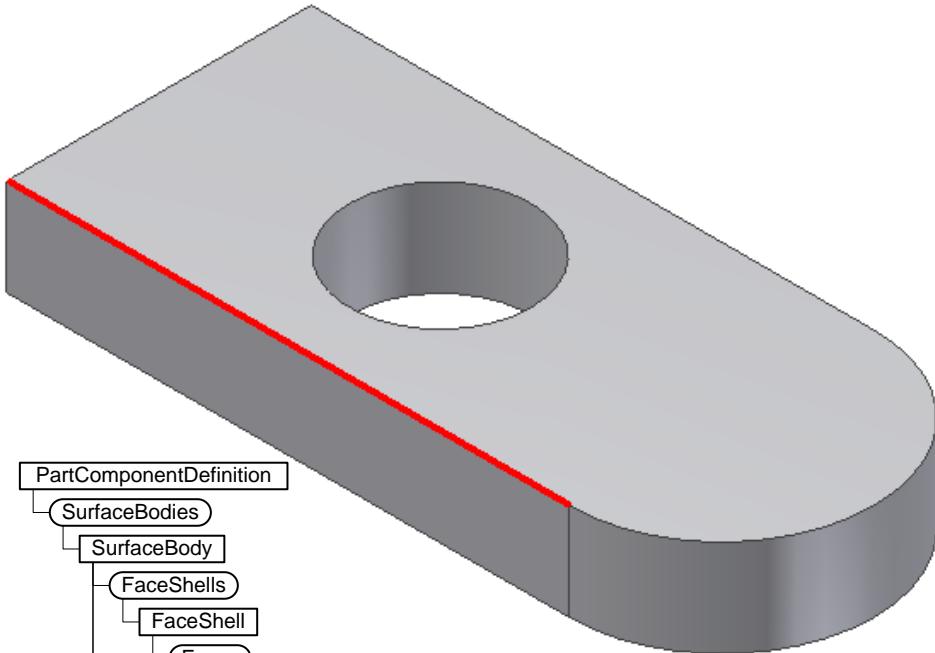
- Face の境界を定義するエッジのセットを指す
- Face 間の接続状況を提供する



# B-Rep

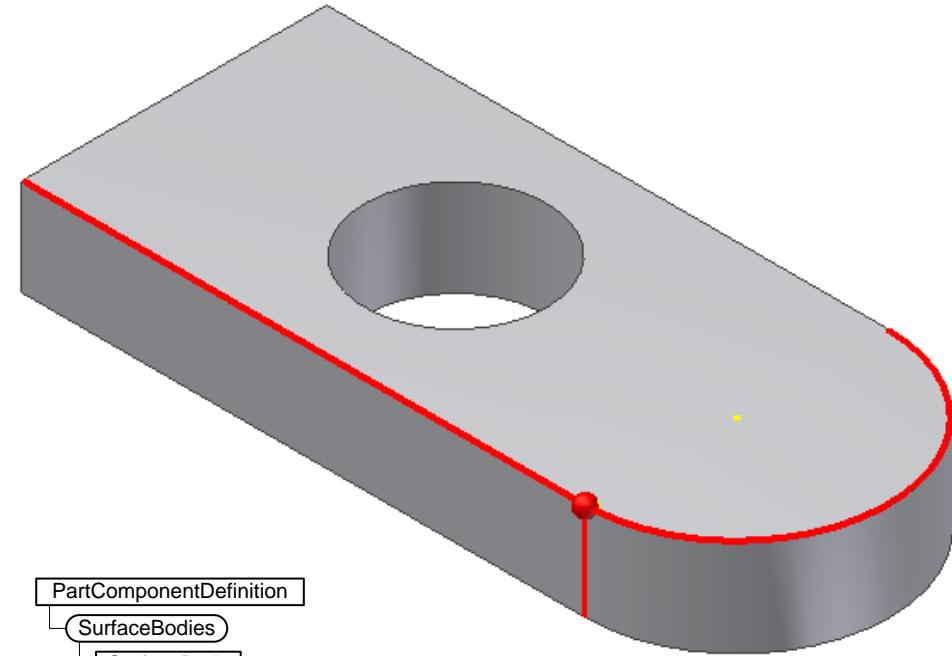
## Edge

- 面の境界の一部を定義する
- 1つの面と他の接続状況を定義

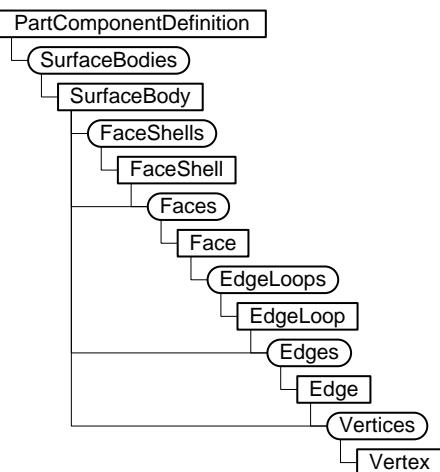


# B-Rep

## Vertex

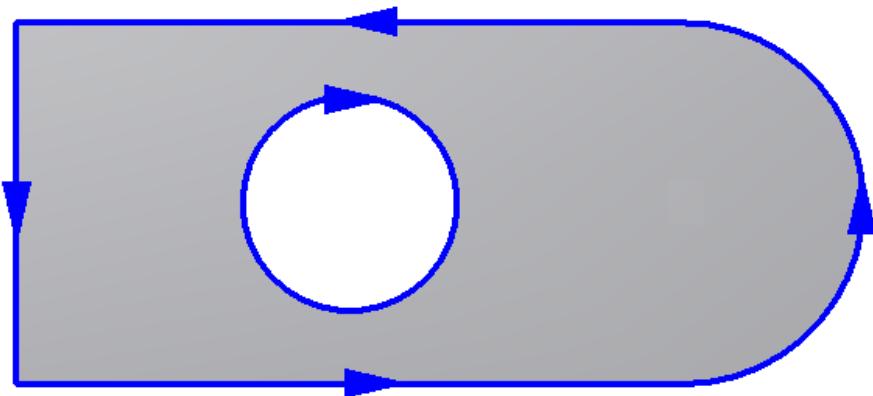


- 面の境界の一部を定義する
- 1つの面と他接続状況を定義

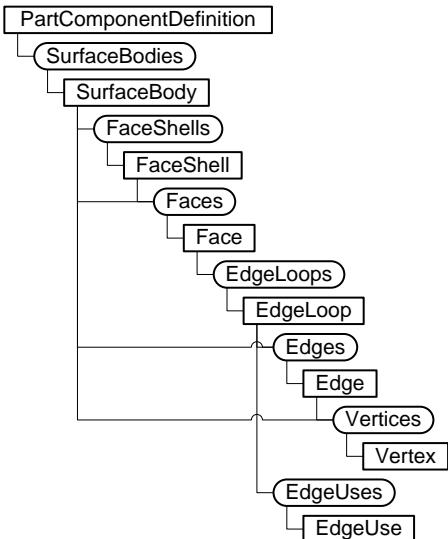


# B-Rep

## EdgeUse



- EdgeUse オブジェクトはFace(面)に境界情報を提供
- サーフェスのパラメータ空間の2Dカーブ
- 重要な循環方向を持つ



# B-Rep

## B-Repアクセス

- *SurfaceBodies*コレクションは、複数の*SurfaceBody*を含む
- B-Rep エンティティへのアクセス方法
  - B-Rep 構造体のトラバース
  - 既存フィーチャーから(*Faces*, *EndFaces*, *StartFaces*, *SideFaces*)
  - 明示された点の近く(*LocateUsingPoint*)
  - 明示されたベクトルの交点 (*FindUsingRay*)
  - エンドユーザーによる選択
  - 以前に割り当てられたアトリビュートに基づいての問い合わせ

# B-Rep

## B-Repアクセス トランザクション (VB.net)

```
Public Sub CountPlanarFacesInEachShell()

    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.ActiveDocument

    ' Iterate through the FaceShell objects.
    Dim iShellCount As Integer
    Dim oShell As FaceShell
    For Each oShell In oPartDoc.ComponentDefinition.SurfaceBodies.Item(1).FaceShells
        iShellCount = iShellCount + 1

        ' Iterate over the faces in this shell.
        Dim iPlanarFaceCount As Integer
        iPlanarFaceCount = 0
        Dim oFace As Face

        For Each oFace In oShell.Faces
            ' Check to see if the face is planar.
            If oFace.SurfaceType = SurfaceTypeEnum.kPlaneSurface Then
                iPlanarFaceCount = iPlanarFaceCount + 1
            End If
        Next

        Debug.Print("FaceShell " & iShellCount &
                   " has " & iPlanarFaceCount & " planar faces.")
    Next

End Sub
```

# B-Rep

## B-Repアクセス 指定点近傍(VB.net)

```
Public Sub GetFaceAtPoint()

    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.ActiveDocument

    Dim oBody As SurfaceBody
    oBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)

    Dim oPoint As Point
    oPoint = _InvApplication.TransientGeometry.CreatePoint(0, 0, 0)

    Dim oFace As Face

    Try
        ' find a face which the point locates on
        oFace = oBody.LocateUsingPoint(ObjectTypeEnum.kFaceObject, oPoint)
        MsgBox("Found face with area " & oFace.Evaluator.Area & " cm^2")
    Catch
        MsgBox("No face at the specified point.")
    End Try

End Sub
```

# B-Rep

## B-Repアクセス 射線方向(VB.net)

```
Public Sub FindUsingRay()

    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.ActiveDocument

    Dim oBody As SurfaceBody
    oBody = oPartDoc.ComponentDefinition.SurfaceBodies.Item(1)

    Dim oTg As TransientGeometry
    oTg = _InvApplication.TransientGeometry

    Dim oFoundEnts As ObjectsEnumerator = Nothing
    Dim oLocPoints As ObjectsEnumerator = Nothing

    ' find the object by a ray
    Call oBody.FindUsingRay(oTg.CreatePoint(-5, 0, 0), _
                           oTg.CreateUnitVector(1, 0, 0), 0.00001, _
                           oFoundEnts, oLocPoints, True)

    If oFoundEnts.Count > 0 Then
        MsgBox("Found " & oFoundEnts.Count & " Entities")
    Else
        MsgBox("No entity along the specified ray.")
    End If

End Sub
```

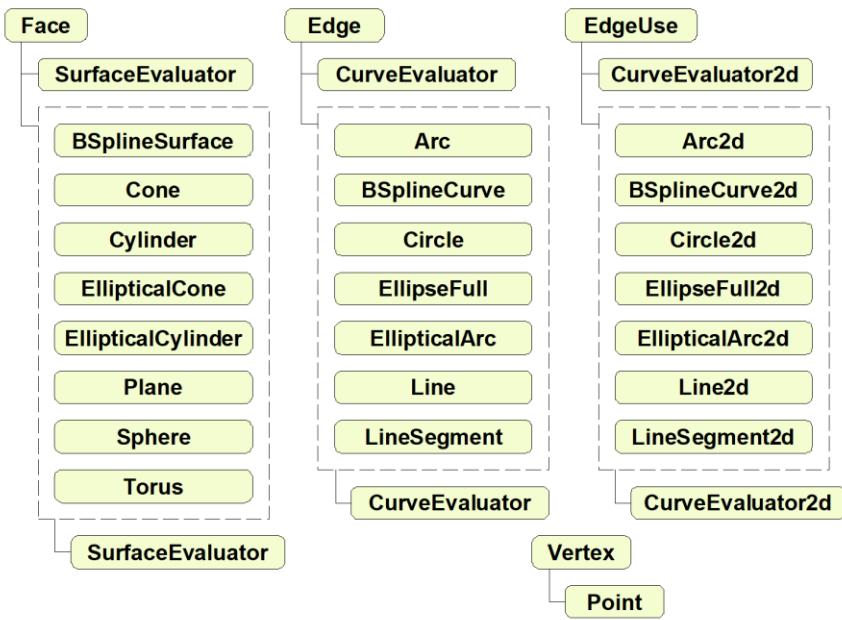
# B-Rep

## B-Repアクセス フィーチャーから

- いくつかのフィーチャーは関連するB-Rep エンティティに直接アクセスするプロパティをサポート
  - ✓ 全フィーチャー
    - *Faces, SurfaceBody*
  - ✓ ExtrudeFeature, RevolveFeature, SweepFeature, LoftFeature, HoleFeature
    - *EndFaces, StartFaces, SideFaces*

# B-Rep

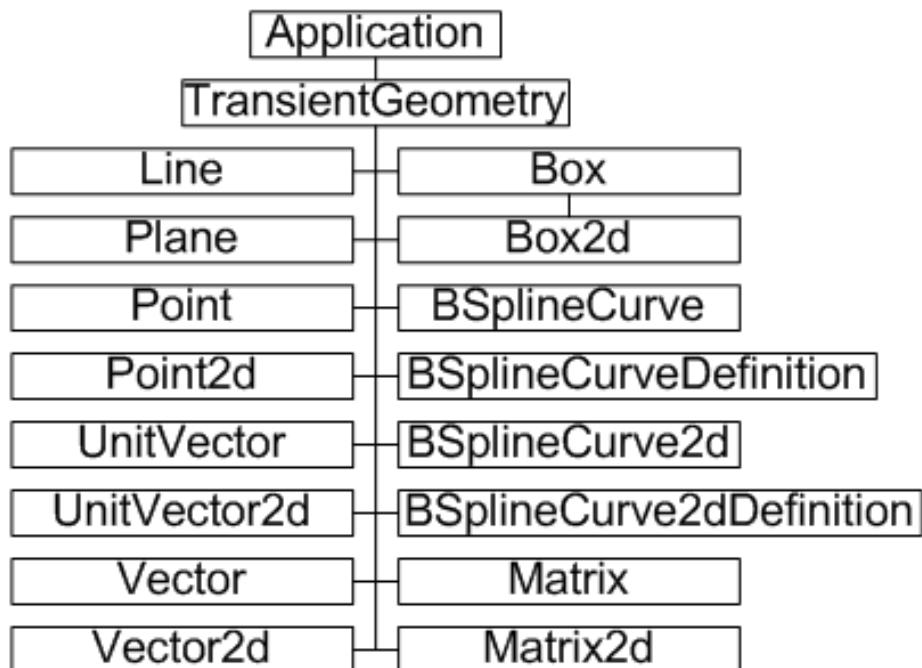
## Transient Geometry



- *Geometry*オブジェクトはトポロジの“スナップショット”。編集する事は可能。しかし得られた元のトポロジに影響はありません
- すべての面は境界をもつ。*Line*および*Line2D*は無限。*LineSegment*および*LineSegment2d*は境界をもつ

# B-Rep

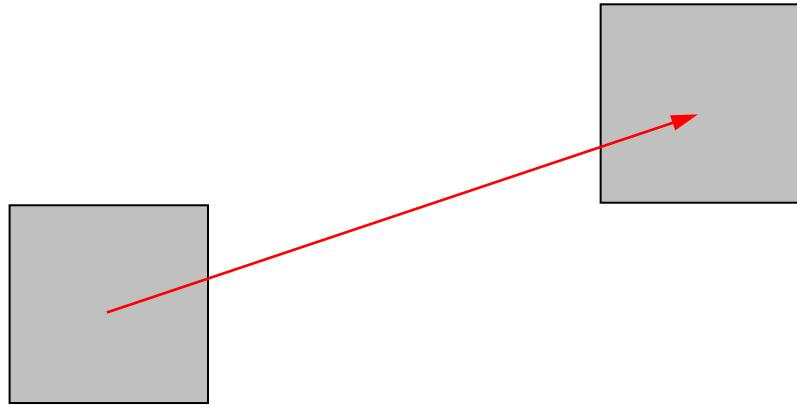
## Transient Geometryの幾何オブジェクト



- *TransientGeometry*オブジェクトにより、メソッドやプロパティの入力および、内部的な計算に使用可能な幾何オブジェクトの作成が可能
  - ✓ `Matrix`, `Matrix2d`
  - ✓ `Vector`, `Vector2d`
  - ✓ `UnitVector`, `UnitVector2d`
  - ✓ `Box`, `Box2d`

# B-Rep

## ベクトル



- ベクトルは方向と大きさを持つ
- ベクトルでパートの移動を表現できる
- 単位ベクトルとは、大きさが1のベクトルで方向を定義する

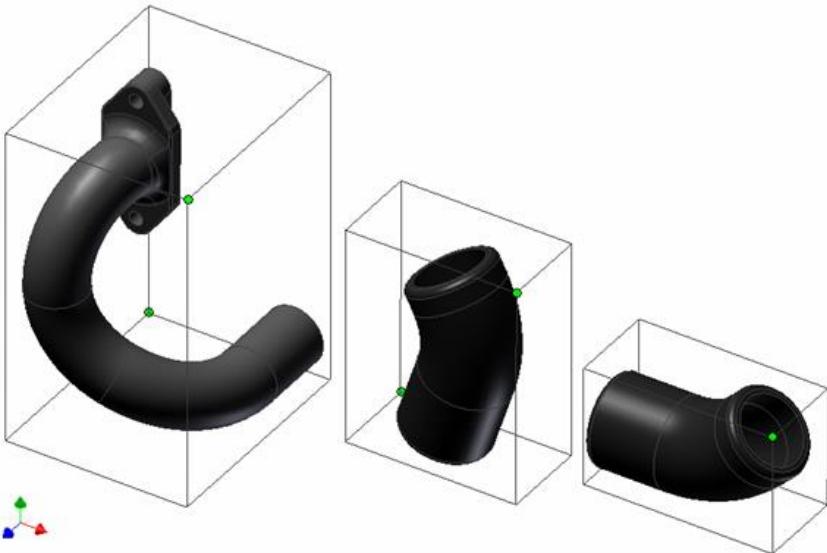
# B-Rep

## ベクトル機能

- 2つのベクトルの比較
  - ✓ *AngleTo*,
  - ✓ *IsEqualTo*
  - ✓ *IsParallelTo*
  - ✓ *IsPerpendicularTo*
  - ...
- ベクトル操作
  - ✓ *AddVector*
  - ✓ *Normalize*
  - ✓ *SubtractVector*
  - ✓ *TransformBy*
  - ✓ *ScaleBy*
  - ...
- 分析
  - ✓ *CrossProduct*
  - ✓ *DotProduct*
  - ✓ *Length*

# B-Rep

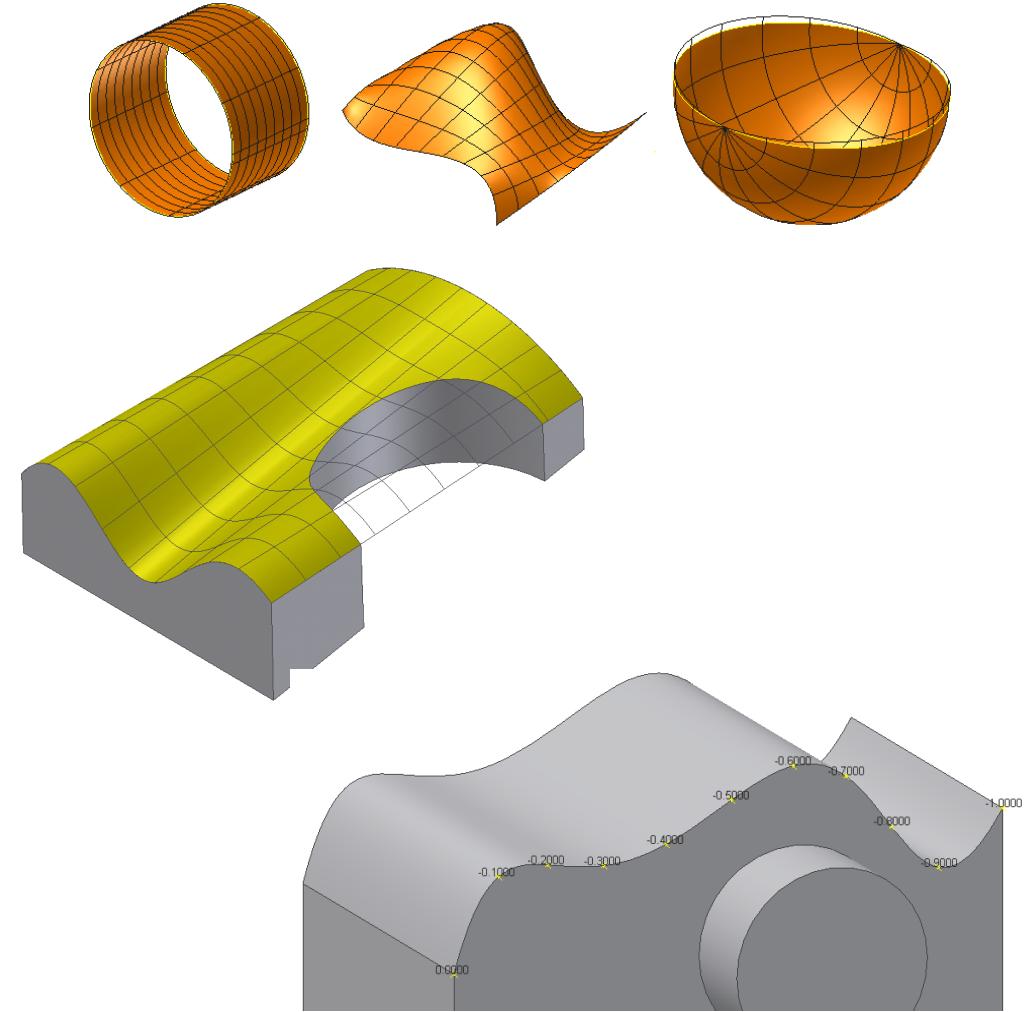
## ボックス(Boxes)



- Boxの最小と最大コーナーを定義した2つの点
- 典型的には、境界Boxのために使われる
- インベンターの境界Boxは、常に座標軸に沿っている
- メソッド
  - ✓ *Contains*
  - ✓ *Expand (dist.)*
  - ✓ *Extend (Point)*

# B-Rep

## Geometry Evaluators



- SurfaceとCurve evaluator オブジェクトは、全てのフェースとカーブの為の一般ジオメトリの問い合わせを提供
  - ✓ Curvatures(曲がり)
  - ✓ Normals (法線)
  - ✓ Parameter space / Model space conversions  
(パラメータ空間 モデル空間 変換)
  - ✓ Tangents (接線)
  - ✓ Parameter Range(パラメータ範囲)
  - ✓ Area (Face) (領域)
  - ✓ Lengths (Curves)(長さ)
  - ✓ Derivatives (派生)

# B-Rep

## Geometry Evaluators サンプル(VB.net)

  ` 入力ベクトルとして同じ方向にある法線の最初のフェースを見つける

```
Public Function FindFace(ByVal oBody As SurfaceBody, ByVal CheckVector As UnitVector) _
    As Face

    FindFace = Nothing

    Dim oFace As Face
    For Each oFace In oBody.Faces
        Dim oSurfEval As SurfaceEvaluator
        oSurfEval = oFace.Evaluator
        ` param rangeを取得
        Dim oParamRange As Box2d
        oParamRange = oSurfEval.ParamRangeRect
        Dim adParams(1) As Double
        adParams(0) = (oParamRange.MinPoint.X + oParamRange.MaxPoint.X) / 2
        adParams(1) = (oParamRange.MinPoint.Y + oParamRange.MaxPoint.Y) / 2
        ` 法線を取得
        Dim adNormals(2) As Double
        Call oSurfEval.GetNormal(adParams, adNormals)
        ` 引数の単位ベクトルと法線の角度をチェック
        Dim oTestVector As UnitVector
        oTestVector = _InvApplication.TransientGeometry.CreateUnitVector( _
            adNormals(0), adNormals(1), adNormals(2))

        If CheckVector.AngleTo(oTestVector) < 0.0001 Then
            FindFace = oFace
            Exit Function
        End If
    Next
End Function
```

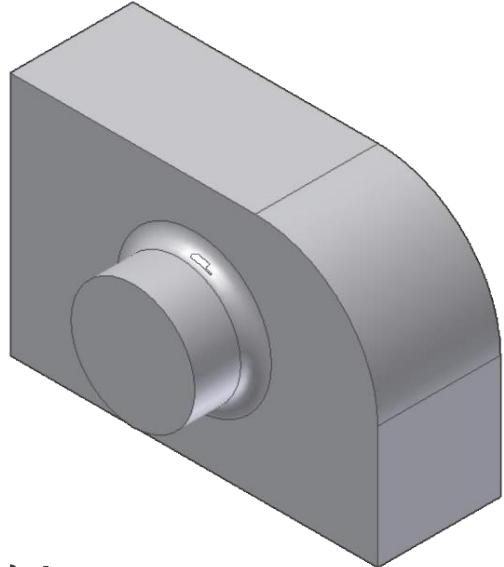
# 演習8

## B-Rep によるアクセスとフィレットの作成

- 演習7で作成した前面のソリッドの表面を見つけ、スケッチを作成することによって、演習を完成してください
- 押し出しを作成し、円筒の“ボス”を作成
- フィレットのためのエッジを見つける
- フィレットの作成

- 演習のヒント

- ✓ フィレットはドキュメントオブジェクトの ComponentDefinition.Features.FilletFeatures.Add Simple で作成できます。

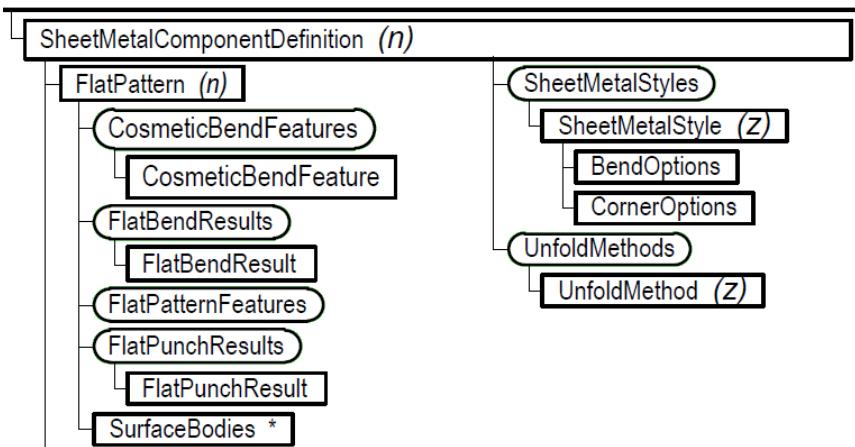




シートメタル

# シートメタル

## シートメタルドキュメント



- シートメタルドキュメントは *PartComponentDefinition* から派生した *SheetMetalComponentDefinition* を持つ *PartDocument*
- すべてのパートフィーチャに加えて、シートメタルに固有の設定であるシートメタルスタイル、シートメタルフィーチャへのアクセスを提供
- モデルの曲げ、フラットパターンへのアクセスを提供
- 折り曲げパンチ、フラットパターンのコスメティックベンド情報へのアクセスを提供

# シートメタル

## シートメタルドキュメントのアクセス、作成サンプル(VB.Net)

```
Public Sub IsSheetMetalPart()

    Dim doc As PartDocument
    doc = _InvApplication.ActiveDocument

    If (doc.SubType = "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}") Then
        MsgBox("ドキュメントはシートメタルです")
    Else
        MsgBox("ドキュメントは通常のパートです")
    End If

End Sub
```

・ テンプレートを使用してシートメタルドキュメントの新規作成

```
Dim oSheetMetalDoc As PartDocument
oSheetMetalDoc = _InvApplication.Documents.Add(DocumentTypeEnum.kPartDocumentObject, _
    _InvApplication.FileManager.GetTemplateFile(_
        DocumentTypeEnum.kPartDocumentObject, _
        SystemOfMeasureEnum.kDefaultSystemOfMeasure, _
        DraftingStandardEnum.kDefault_DraftingStandard, _
        "{9C464203-9BAE-11D3-8BAD-0060B0CE6BB4}))
```

# シートメタル

## シートメタルスタイル

SheetMetalComponentDefinition

SheetMetalStyles

SheetMetalStyle

BendOptions

CornerOptions

UnfoldMethods

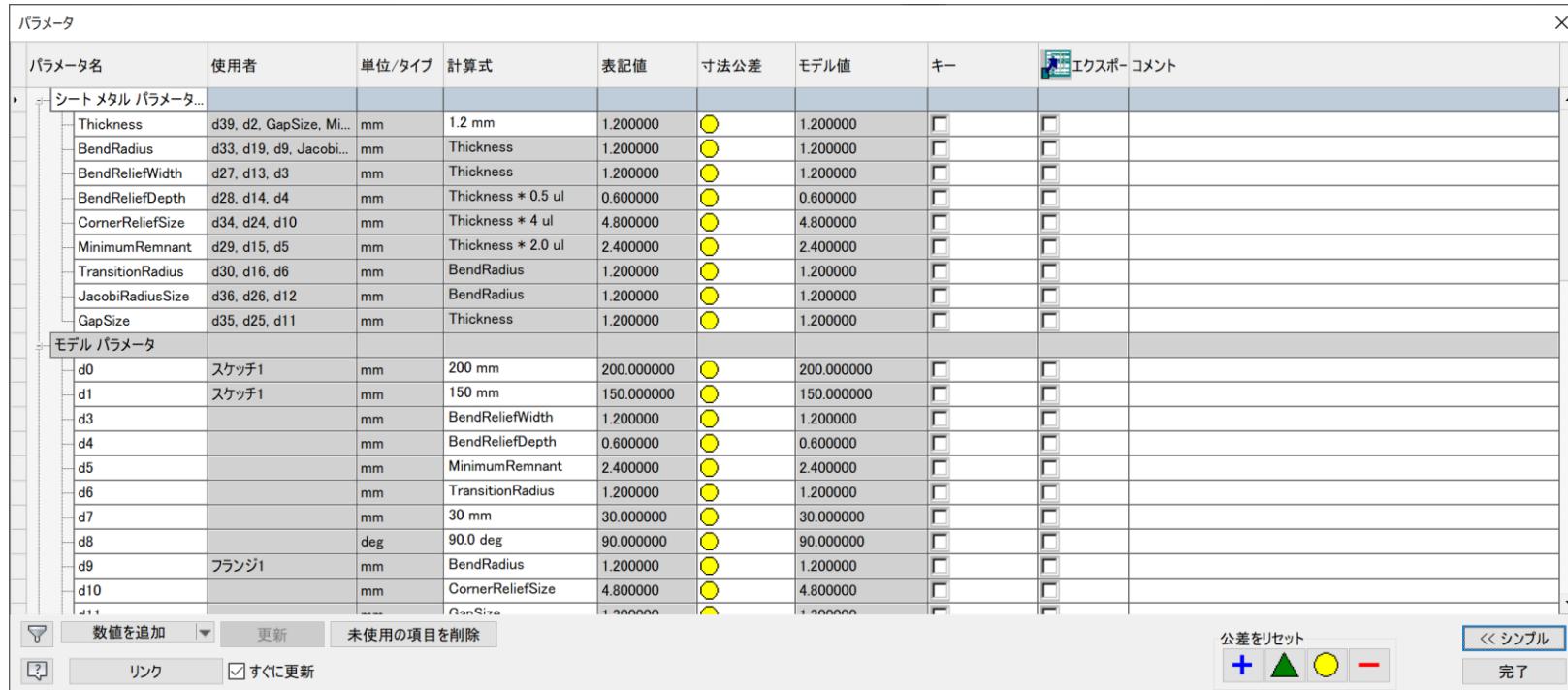
UnfoldMethod

- *SheetMetalStyles*と*UnfoldMethods*オブジェクトにより様々な設定の保存や定義が可能
- シートメタルスタイルの有効化することで、シートメタルドキュメントに適用されるスタイルを定義
- これらの値は、*SheetMetalStyle* オブジェクトのプロパティでアクセスされる
  - ✓ *BendRadius*, *BendReliefDepth*, *BendReliefWidth*, *CornerReliefSize*, *GapSize*, *Thickness*, ...
- これらプロパティは文字列により取得・設定をする

# シートメタル

## シートメタルスタイルとパラメータ

- 多くの*SheetMetalStyle*プロパティはリファレンスパラメータの設定に使用される。そのため、これらのパラメータはスタイル値により定義される読み取り専用パラメータとなる

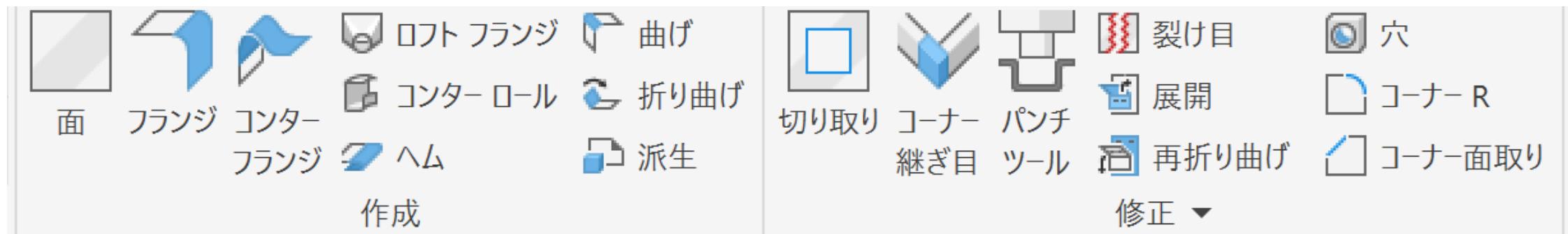


- これらのパラメータは、*SheetMetalComponentDefinition*プロパティから直接アクセス可能

# シートメタル

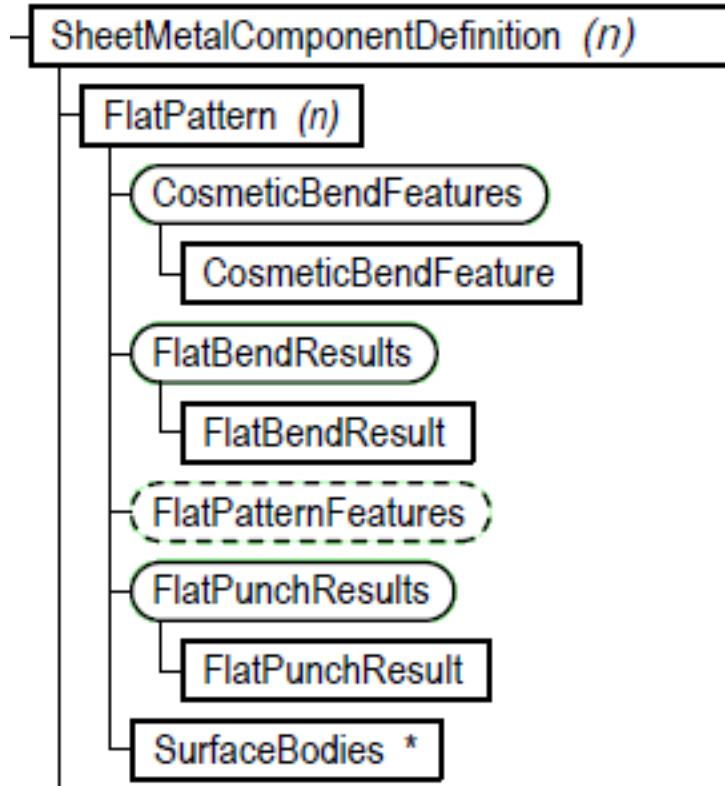
## シートメタルフィーチャー

- *SheetMetalComponentDefinition.Features* プロパティにより *SheetMetalFeatures* オブジェクトを取得可能。*SheetMetalFeatures* は *PartFeatures* から派生
- *SheetMetalFeatures* オブジェクトは様々なシートメタルフィーチャのサポートを提供
- すべてのシートメタルフィーチャの取得が可能。また多くのシートメタルフィーチャの作成をサポート
- すべてのシートメタルフィーチャは“definition” コンセプトを使用し、フィーチャ種別それぞれに対して、フィーチャの情報を定義する、definitionオブジェクトが存在する



# シートメタル

## シートメタルフラットパターンへのアクセス



- フラットパターンの存在確認
  - ✓ `SheetMetalComponentDefinition.HasFlatPattern`
- フラットパターンの作成
  - ✓ `SheetMetalComponentDefinition.Unfold` および `Unfold2`
- フラットパターンの削除
  - ✓ `FlatPattern.Delete` および `FlatPattern.DeleteObjects`
- ユーザインターフェイスで編集モードの開始と終了
  - ✓ `FlatPattern.Edit`, `FlatPattern.ExitEdit`
- フラットパターンの方向の制御
  - ✓ `FlatPattern.FlipBaseFace`
  - ✓ `FlatPattern.GetAlignment`,
  - ✓ `FlatPattern.SetAlignment`
- フラットパターン環境でのモデリングサポート
  - ✓ `Features`、`Sketches`、`WorkFeatures`へのアクセス

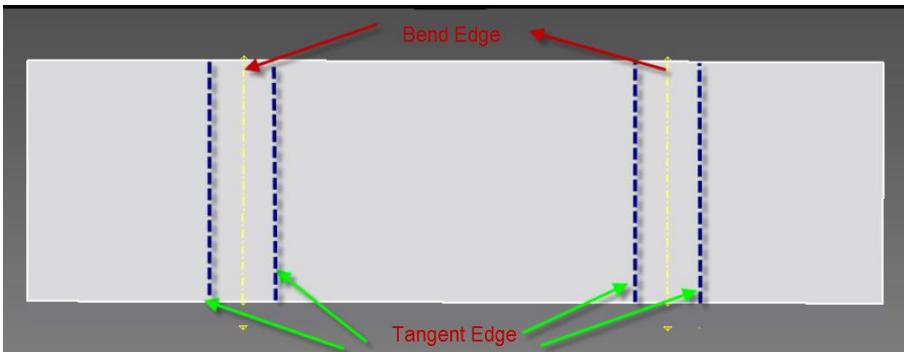
# シートメタル

## フラットモデルへのアクセス

- フラットパターンのB-Repへのアクセス
  - ✓ *FlatPattern.SurfaceBody*
- フラットパターンのエクステントの取得
  - ✓ *FlatPattern.Length*, *FlatPattern.Width*
- フラットパターンのTopおよびBottomフェースの取得
  - ✓ *FlatPattern.TopFace*, *FlatPattern.BottomFace*

# シートメタル

## フラットパターンの折り曲げ情報



- フラットパターンから、曲げ・展開線を取得。引数EdgeTypeにより取得したい線種を指定
  - ✓ *FlatPattern.GetEdgesOfType( \_  
EdgeType As FlatPatternEdgeTypeEnum, \_  
[TopFaceEdges As Boolean = True]) As Edges*
- 曲げおよび接線Edgeオブジェクトとして表現される。これらのEdgeオブジェクトはwireframeエッジを表し、サーフェースやソリッドとは関連していない。
- *FlatPattern.FlatBendResults*は曲げの情報へのアクセスを提供

# シートメタル

## フラットパターンのPunch情報

- *FlatPattern.PunchRepresentationType* プロパティにより、punch featuresフィーチャーの表示のコントロールが可能。*Punches*によりPunchを表すスケッチを定義することが可能。またモデル内のすべてのPunchesに対して、ソリッドまたはスケッチ表現のどちらを見せるかを選択可能
- *FlatPattern.FlatPunchResults* プロパティはモデル内のpunch情報(位置、角度、方向、関連するエッジやフェース、その他)へのアクセスを提供。曲げ、展開モデル内に作られた、Punchフィーチャーを含む、Punchのソリッド及びスケッチの両方をサポート

# シートメタル

## フラットパターンのExport

- フラットシートメタルパートは、DATAIOオブジェクトを使ってDWG・DXFを書き出すことが可能。
- オプション:
  - ✓ TangentLayer= "IV\_TANGENT"
  - ✓ BendLayer = "IV\_BEND"
  - ✓ ToolCenterLayer= "IV\_TOOL\_CENTER"
  - ✓ ArcCentersLayer= "IV\_ARC\_CENTERS"
  - ✓ OuterProfileLayer= "IV\_OUTER\_PROFILE"
  - ✓ FeatureProfilesLayer= "IV\_FEATURE\_PROFILES"
  - ✓ InteriorProfilesLayer= "IV\_INTERIOR\_PROFILES"
  - ✓ AcadVersion = "2000"(Can be "R12", "R13", "R14", or "2000")(R12はDXFのみサポート)

# シートメタル

## フラットパターンのDXF Export サンプル(VB.net)

```
Public Sub WriteSheetMetalDXF()

    ' アクティブドキュメントを取得。パートドキュメントと想定
    Dim oDoc As PartDocument
    oDoc = _InvApplication.ActiveDocument

    ' DataIO オブジェクトを取得
    Dim oDataIO As DataIO
    oDataIO = oDoc.ComponentDefinition.DataIO

    ' DXFファイル出力定義をする文字列を作成
    Dim sOut As String
    sOut = "FLAT PATTERN DXF?AcadVersion=R12&OuterProfileLayer=Outer"

    ' DXFファイルを出力
    oDataIO.WriteDataToFile(sOut, "c:\Temp\flat.dxf")

End Sub
```



アセンブリ



# アセンブリ

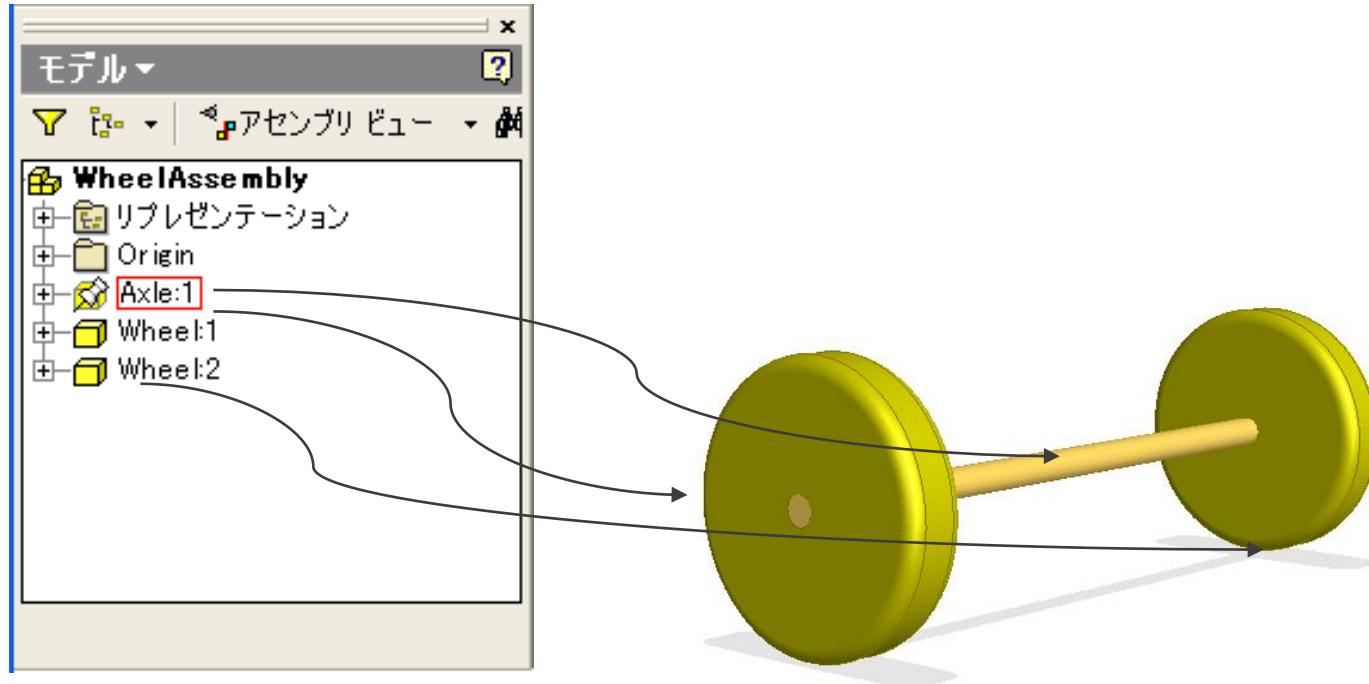
## アセンブリドキュメント



- APIは、アセンブリ機能のほとんどをサポート
  - ✓ コンポーネントの配置、作成
  - ✓ コンポーネント編集
  - ✓ パターン
  - ✓ 拘束
  - ✓ ワークフィーチャー
  - ✓ パラメータ
  - ✓ iMate
  - ✓ スケッチ
  - ✓ フィーチャー
  - ✓ リプレゼンテーション
  - ✓ iAssembly
  - ✓ BOM

# アセンブリ

GUIから見たアセンブリドキュメント



# アセンブリ

## APIから見たアセンブリドキュメント

### WheelAssembly.iam

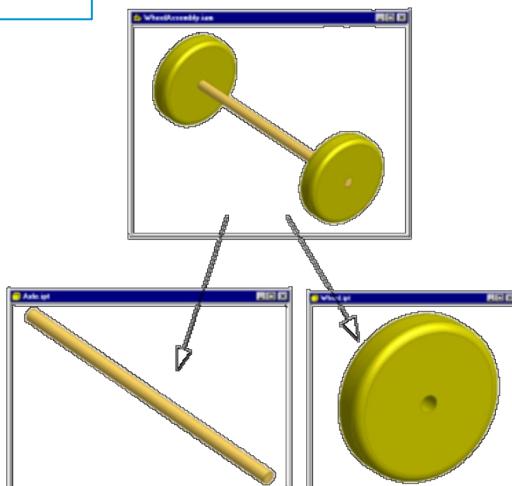
#### References:

1. Axle.ipt
2. Wheel.upt

#### Occurrences:

1. Axle:1, Reference1, (0,0,0,...), Visible, ...
2. Wheel:1, Reference2, (0,0,-2,...), Visible, ...
3. Wheel:2 Reference3, (0,0,-2,...), Visible, ...

- アセンブリドキュメントは以下ののような情報を持つ
  - ✓ 他のドキュメントの参照
  - ✓ オカレンス情報、拘束
  - ✓ ワークフィーチャー
- アセンブリドキュメントは、ジオメトリ情報を持たず、唯一パートと他のアセンブリドキュメントへの参照のみを持つ(アセンブリフィーチャーは、特殊な例外)



# アセンブリ

## アセンブリドキュメントAPI構造

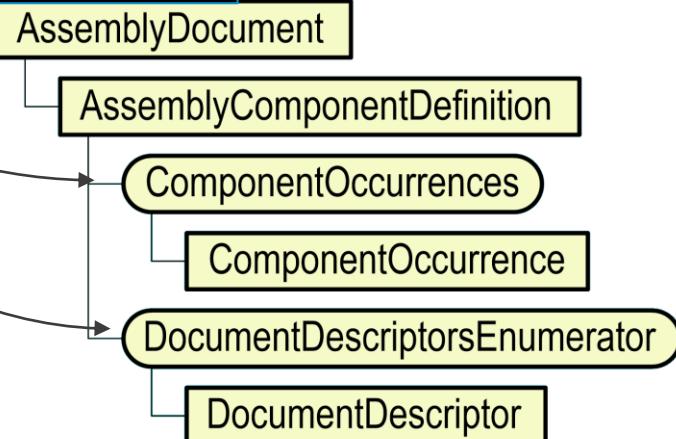
### WheelAssembly.iam

#### References:

1. Axle.ipt
2. Wheel.upt

#### Occurrences:

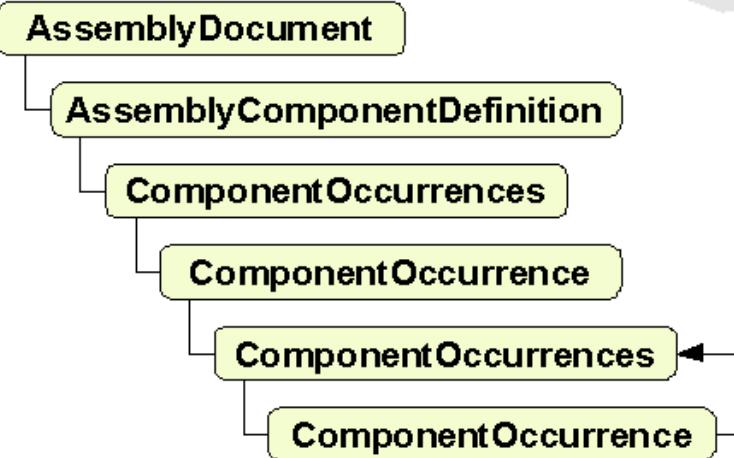
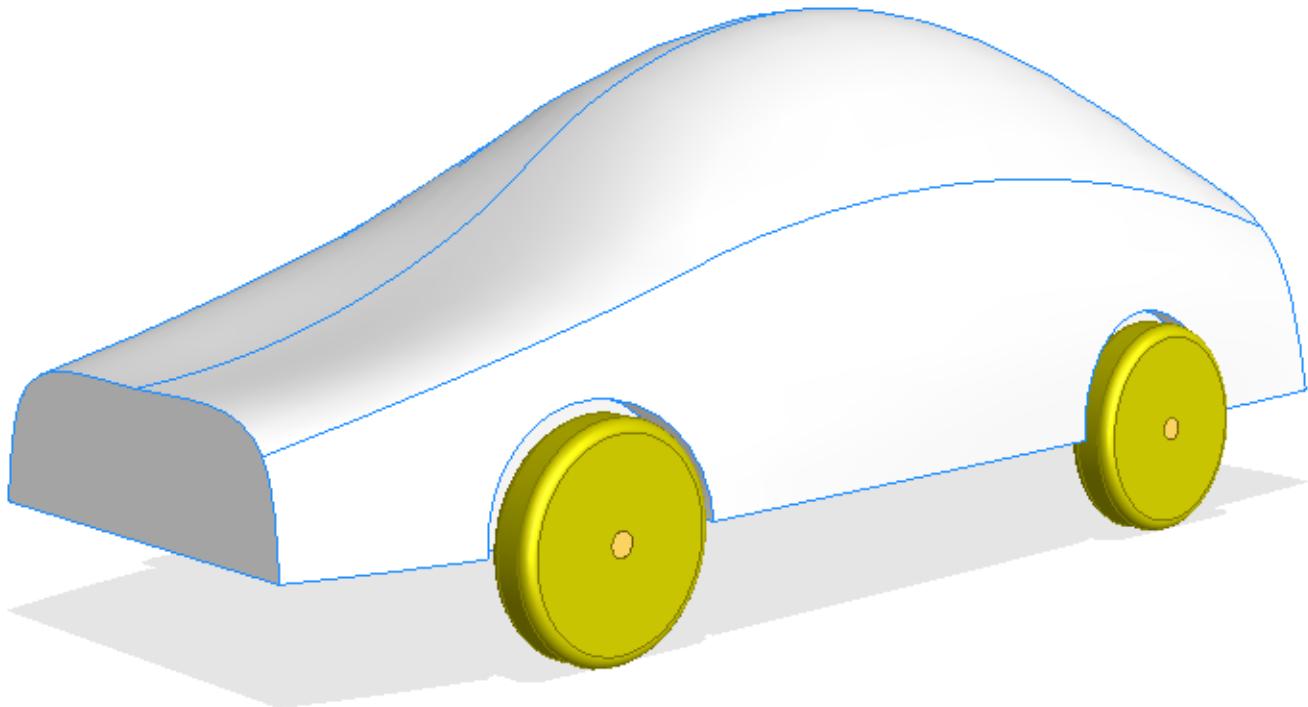
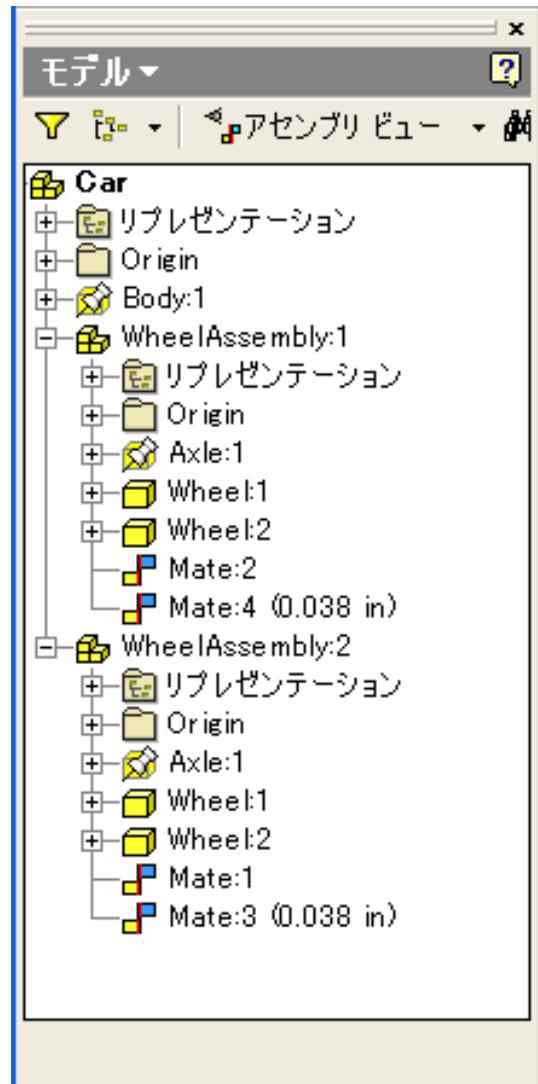
1. Axle:1, Reference1, (0,0,0,...), Visible, ...
2. Wheel:1, Reference2, (0,0,-2,...), Visible, ...
3. Wheel:2 Reference3, (0,0,-2,...), Visible, ...



- *ComponentOccurrences*コレクションオブジェクトは、Occurrencesプロパティを通じてアクセスでき、既存のオカレンスへの繰り返しアクセスを提供。また、オカレンスの追加も可能
- *DocumentDescriptorsEnumerator*オブジェクトは、ReferencedDocumentDescriptorsプロパティを通じてアクセスでき、アセンブリが参照しているドキュメントへのアクセスを提供

# アセンブリ

## アセンブリ構造の走査



# アセンブリ

## アセンブリ構造の走査のサンプル(VB.Net)

```
Public Sub AssemblyTraversal()

    ' アクティブなドキュメントを取得。アセンブリであると想定
    Dim oAsmDoc As AssemblyDocument
    oAsmDoc = _InvApplication.ActiveDocument

    Call TraverseAsm(oAsmDoc.ComponentDefinition.Occurrences, 1)

End Sub

Private Sub TraverseAsm(ByVal oOccurrences As ComponentOccurrences, ByVal Level As Integer)

    ' 配下のオカレンスに対して繰り返す
    Dim oOcc As ComponentOccurrence
    For Each oOcc In oOccurrences
        ' オカレンス名を出力
        Debug.Print(Space(Level * 3) & oOcc.Name)

        ' オカレンスがアセンブリの場合、配下のオカレンスに対して再起呼び出し
        If oOcc.DefinitionDocumentType = DocumentTypeEnum.kAssemblyDocumentObject Then
            Call TraverseAsm(oOcc.SubOccurrences, Level + 1)
        End If
    Next

End Sub
```

# アセンブリ

## オカレンスの作成

- オカレンス作成API

- ✓ *Add( FileName As String, Position As Matrix ) As ComponentOccurrence*
- ✓ *AddByComponentDefinition( CompDef As ComponentDefinition, Position As Matrix ) As ComponentOccurrence*
- ✓ *AddUsingiMates( FileName As String, Position As Matrix ) As ComponentOccurrence*
- ✓ *AddCustomiPartMember( FactoryFileName As String,  
Position As Matrix, FullFileName As String, [Row],  
[CustomInput] ) As ComponentOccurrence*
- ✓ *AddiPartMember( FactoryFileName As String, Position As Matrix, [Row] ) As  
ComponentOccurrence*
- ✓ *AddiAssemblyMember( FactoryDocumentName As String,  
Position As Matrix, [Row], [Options] ) As ComponentOccurrence*

# アセンブリ

## オカレンスの作成サンプル(VB.Net)

```
Public Sub AddFromFile()

    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument
    ' 既存のバーツファイルをオカレンスとしてアセンブリに配置
    Dim oMatrix As Matrix
    oMatrix = _InvApplication.TransientGeometry.CreateMatrix

    Dim oOcc As ComponentOccurrence
    oOcc = oDoc.ComponentDefinition.Occurrences.Add("C:\Temp\Part1.ipt", oMatrix)

End Sub

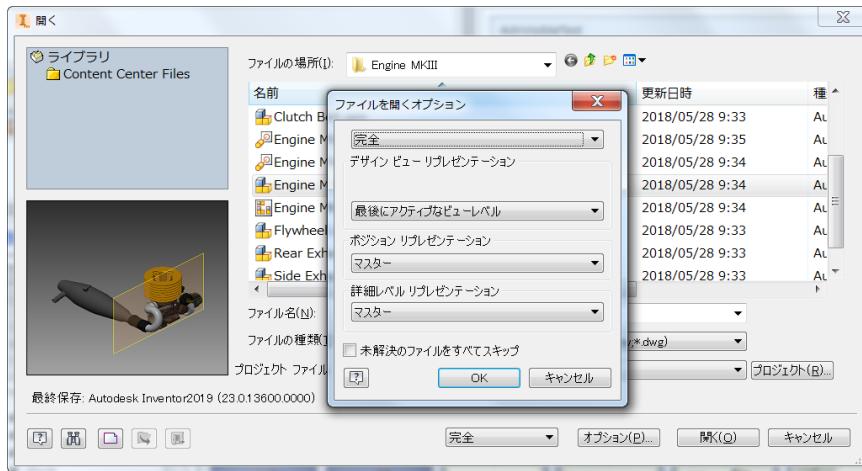
Public Sub AddFromMemory()

    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument
    '新規バーツを作成
    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.Documents.Add(kPartDocumentObject, False)

    Dim oMatrix As Matrix
    oMatrix = _InvApplication.TransientGeometry.CreateMatrix
    '新規作成したバーツをオカレンスとして配置
    Dim oOcc As ComponentOccurrence
    oOcc = oDoc.ComponentDefinition.Occurrences.AddByComponentDefinition( _
        oPartDoc.ComponentDefinition, oMatrix)

End Sub
```

# アセンブリ オカレンス作成のオプション



- *AddWithOptions(FullDocumentName As String, Position As Matrix, Options As NameValueMap) As ComponentOccurrence*
- オプション設定
  - ✓ PrivateRepresentationFileName
  - ✓ DesignViewRepresentation
  - ✓ PositionalRepresentation
  - ✓ LevelOfDetailRepresentation
  - ✓ UseiMate
  - ✓ DesignViewAssociative

# アセンブリ

# オカレンス作成のオプション サンプル(VB.Net)

# アセンブリ

## Matrixとは？

- マトリクスは数学の行列
  - ✓ 3-D マトリクスは4x4 マトリクス

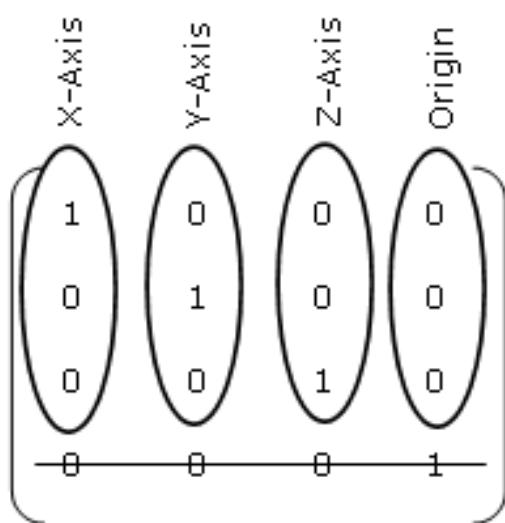
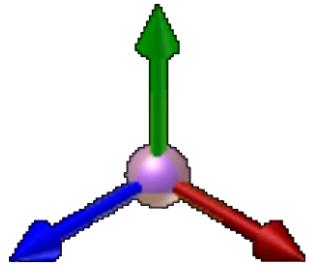
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- ✓ 2-D マトリクスは3x3 マトリクス

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

# アセンブリ

## Inventorでのマトリクス

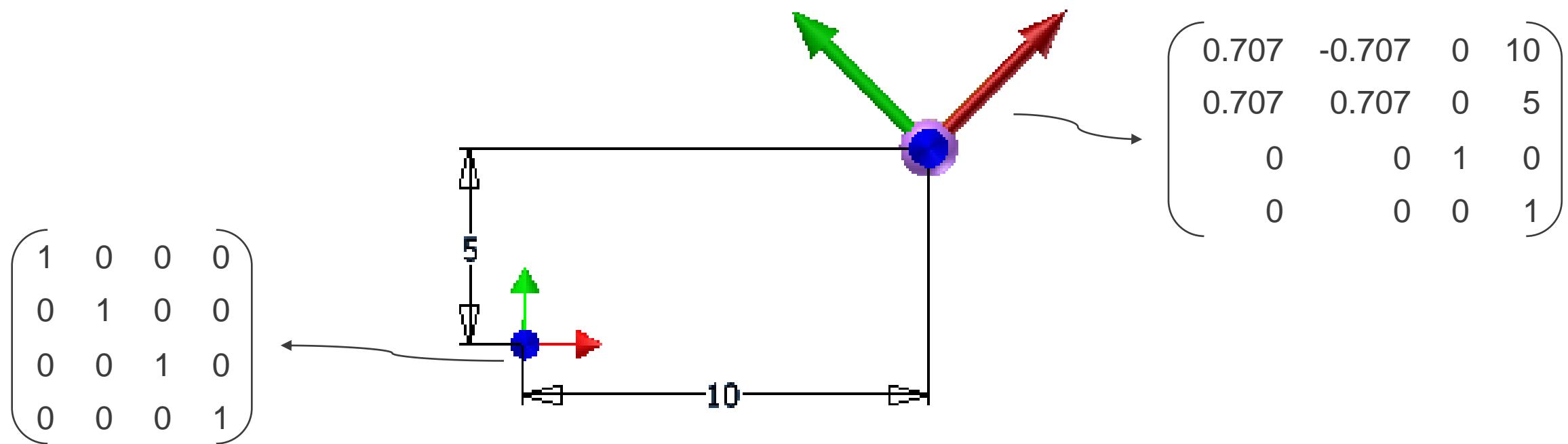


- コンピュータグラフィクスでは、通常行列は以下の用途で使用
  - ✓ 座標システム定義
  - ✓ 変換定義
- でInventorではアセンブリのオカレンス、パートのスケッチ、図面のビューでこのコンセプトを使用

# アセンブリ

## 座標システム上のマトリクス

- 単位行列は、ベースの座標系と同じで、ある座標系を定義します（変換ではない）



# アセンブリ

## マトリクスとオカレンス

- オカレンスを配置する時に、マトリクスはアセンブリ内のパート位置を定義する。アセンブリ空間内の、パート座標系の位置を定義
- オカレンスの *Transformation* プロパティは、オカレンスのアセンブリ内の現在位置を定義するマトリクスを返す
- オカレンスの *Transformation* プロパティへの設定は、(拘束を考慮した)オカレンスの再配置をする
- *SetTransformWithoutConstraints*は、拘束も無視して、(アセンブリの次回再計算までの間)オカレンスを変形する

# アセンブリ

## 変換としてのマトリクス

- マトリクスは、既存オブジェクトの変形に使うことが可能
  - ✓ アセンブリ内のオカレンスの再配置
  - ✓ 一方の座標系から他方の座標系への変形。例えば、アセンブリ内である点を、あるパートから他のパートへ変形
- 変形のためのマトリクスは、変更する差分で定義。変更は移動や回転が可能

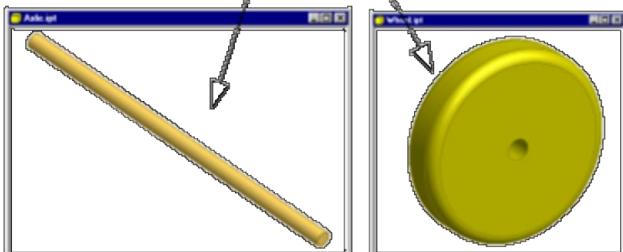
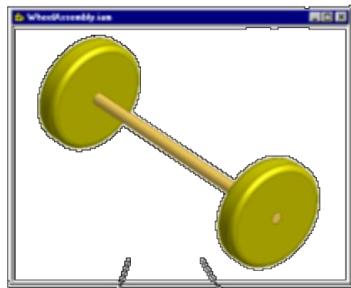
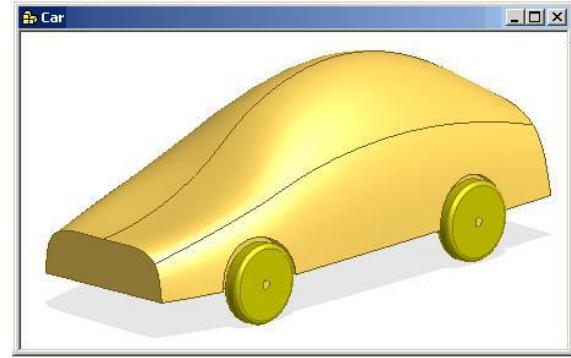
# アセンブリ

## マトリクスの機能

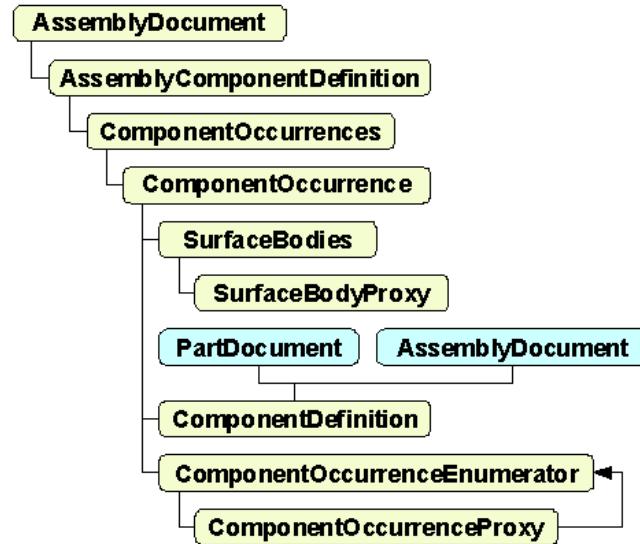
- *Matrix.Invert* ……マトリクスが定義する変形を逆にする
- *Matrix.TransformBy* ……第2のマトリクスによる変形定義を含むマトリクスに変更
- *Matrix.Cell* ……マトリクスの個々のセルのGet/Set
- *SetCoordinateSystem, SetToAlignCoordinateSystems, SetToIdentity, SetToRotateTo, SetToRotation, SetTranslation* は、マトリクス定義を便利にする

# アセンブリ

## アセンブリドキュメントプロキシ



- 問: アセンブリにはジオメトリが存在しないので、どのようにアセンブリ内のジオメトリにアクセスするか
- 答え: プロキシは、あたかもエンティティがアセンブリで実際に存在するかのように、エンティティを表します

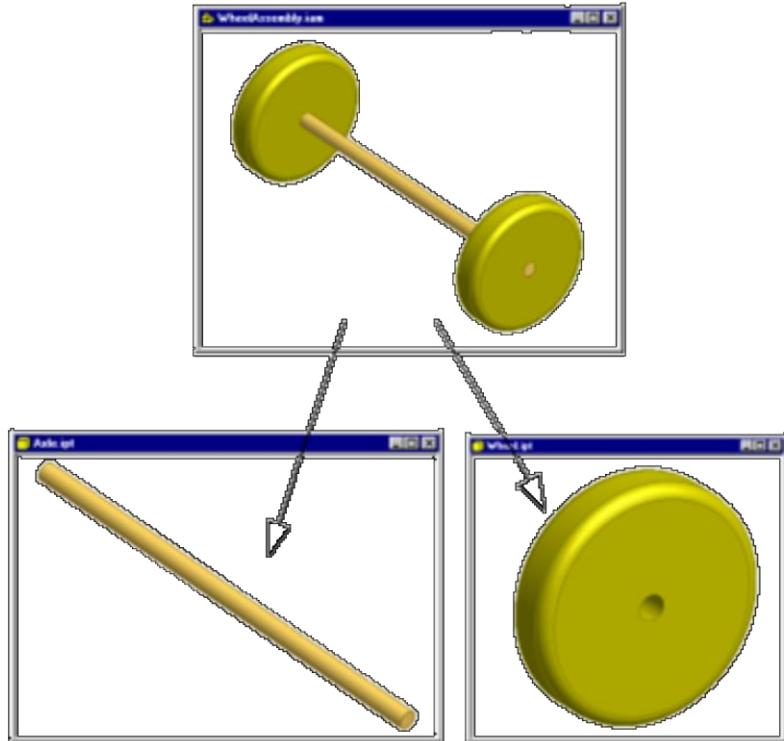


# アセンブリ

## プロキシオブジェクト

- プロキシオブジェクトは、レギュラーオブジェクトから派生
  - ✓ オリジナルオブジェクトがサポートする全てのメソッドとプロパティをサポート
  - ✓ これらのメソッドとプロパティは、アセンブリコンテキストでの情報を返す
- ベースクラスオブジェクトの機能に加え、プロキシは以下をサポート
  - ✓ *ContainingOccurrence*…そのプロキシを保持しているオカレンスを取得
  - ✓ *NativeObject*…プロキシが表す実際のオブジェクトを取得

# アセンブリ プロキシオブジェクト



- プロキシは実際のオブジェクトのパスを定義
  - Cylindrical Face 1  
Wheel:1\CylinderFace
  - Cylindrical Face 2  
Wheel:2\CylinderFace
- プロキシはユーザーがエンティティを選択した際に取得
- プロキシは *CreateGeometryProxy* メソッドを使って作成可能
- 既存のプロキシパスは *AdjustProxyContext* を使って調整が可能
- パスは *OccurrencePath* プロパティで調査可能

# アセンブリ

## プロキシ作成サンプル(VB.Net)

```
Public Sub CreateProxy()

    Dim oAsmDef As AssemblyComponentDefinition
    oAsmDef = _InvApplication.ActiveDocument.ComponentDefinition

    Dim oOcc1 As ComponentOccurrence = oAsmDefOccurrences(1)
    Dim oOcc2 As ComponentOccurrence = oAsmDefOccurrences(2)

    'オカレンスを通してVertexVertexProxyオブジェクトを取得
    Dim oVertexPx1 As VertexProxy
    oVertexPx1 = oOcc1.SurfaceBodies(1).Vertices(1)

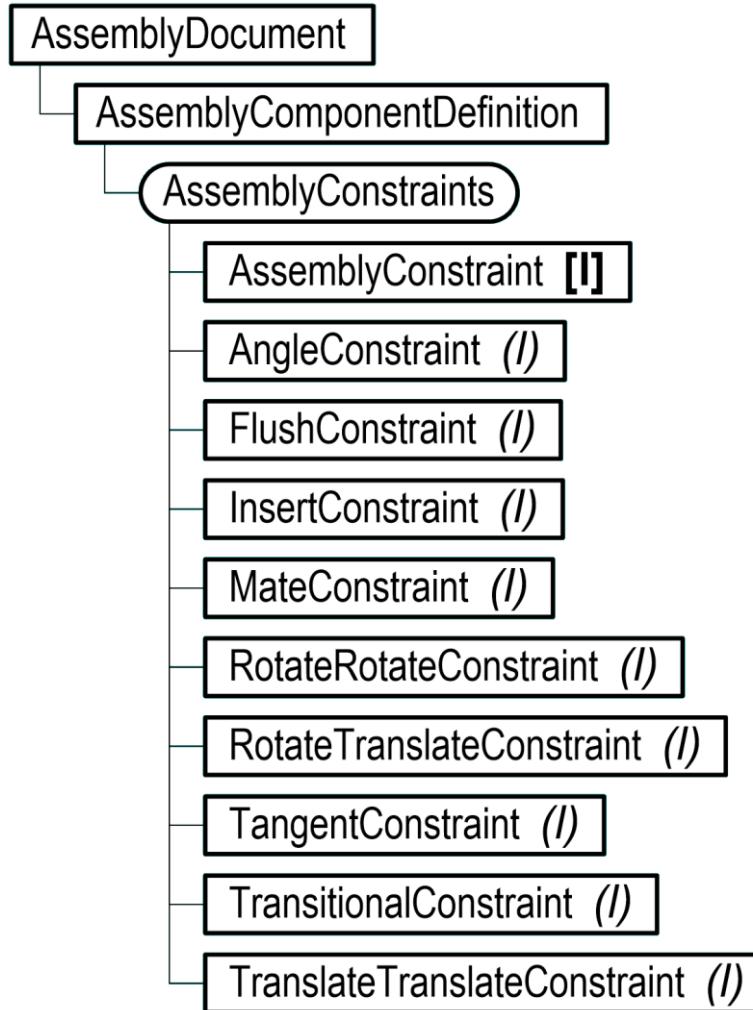
    ' パーツから頂点を取得し、VertexProxyオブジェクトを作成
    Dim oVertex2 As Vertex
    oVertex2 = oOcc2.Definition.SurfaceBodies(1).Vertices(1)

    Dim oVertexPx2 As VertexProxy = Nothing
    Call oOcc2.CreateGeometryProxy(oVertex2, oVertexPx2)

End Sub
```

# アセンブリ

## 拘束



- 拘束の作成は、アセンブリのワークジオメトリまたは、アタッチしたパートのエンティティのプロキシを指定可能
- 拘束の参照は、拘束が関連しているエンティティと、拘束を管理しているパラメータを返す

# アセンブリ

## 拘束の作成サンプル(VB.net) Nativeオブジェクトから作成

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

    ' 1番目と2番目のオカレンスを取得
    Dim oOcc1 As ComponentOccurrence
    oOcc1 = oAsmCompDefOccurrences.Item(1)

    Dim oOcc2 As ComponentOccurrence
    oOcc2 = oAsmCompDefOccurrences.Item(2)

    ' それぞれのオカレンスのXY平面を取得。
    ' 取得した作業平面は、パートのコンテキストとして取得される
    Dim oPartPlane1 As WorkPlane
    oPartPlane1 = oOcc1.Definition.WorkPlanes.Item(3)

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

    ' アセンブリのコンテキストでの作業平面が必要なため、作業平面にプロキシとして作成する必要がある。
    ' プロキシは、アセンブリコンテキストでの作業平面を表す
    Dim oAsmPlane1 As WorkPlaneProxy
    Call oOcc1.CreateGeometryProxy(oPartPlane1, oAsmPlane1)

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

    ' 作業平面プロキシを用いて拘束を作成
    Call oAsmCompDef.Constraints.AddMateConstraint(oAsmPlane1, oAsmPlane2, 0)
End Sub
```

# アセンブリ

## 拘束の作成サンプル(VB.net) プロキシオブジェクトから作成

```
Public Sub MateConstraintWithLimits()

    ' アセンブリコンポーネントへの参照を取得
    Dim oAsmCompDef As AssemblyComponentDefinition
    oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition

    ' select setの参照を取得。
    Dim oSelectSet As SelectSet
    oSelectSet = ThisApplication.ActiveDocument.SelectSet

    ' 選択セットを検証
    If oSelectSet.Count <> 2 Then
        MsgBox ("Mateに適切な2つのエンティティを選択してください")
        Exit Sub
    End If

    ' 選択した2つのエンティティを取得
    Dim oBrepEnt1 As Object
    Dim oBrepEnt2 As Object
    oBrepEnt1 = oSelectSet.Item(1)
    oBrepEnt2 = oSelectSet.Item(2)

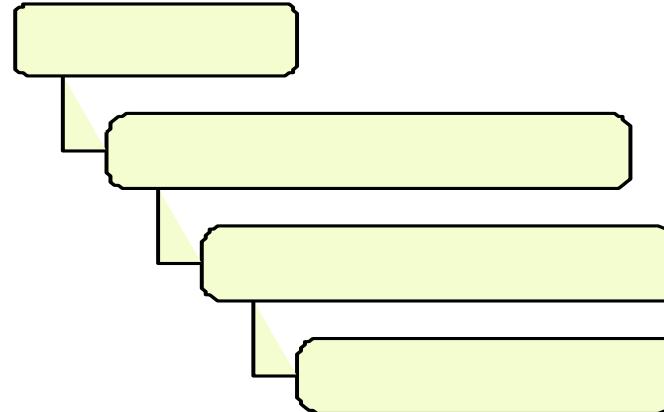
    ' 2つのパーツ間に、オフセット値0のMate拘束を作成。
    Dim oMate As MateConstraint
    oMate = oAsmCompDef.Constraints.AddMateConstraint(oBrepEnt1, oBrepEnt2, 0)

    ' 最大値に2インチを設定
    oMate.ConstraintLimits.MaximumEnabled = True
    oMate.ConstraintLimits.Maximum.Expression = "2 in"

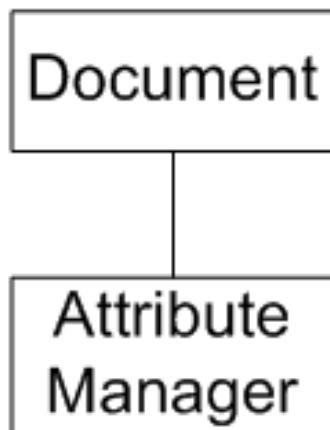
    ' 最小値に-2インチを設定
    oMate.ConstraintLimits.MinimumEnabled = True
    oMate.ConstraintLimits.Minimum.Expression = "-2 in"
End Sub
```

# アセンブリ

## アトリビュート

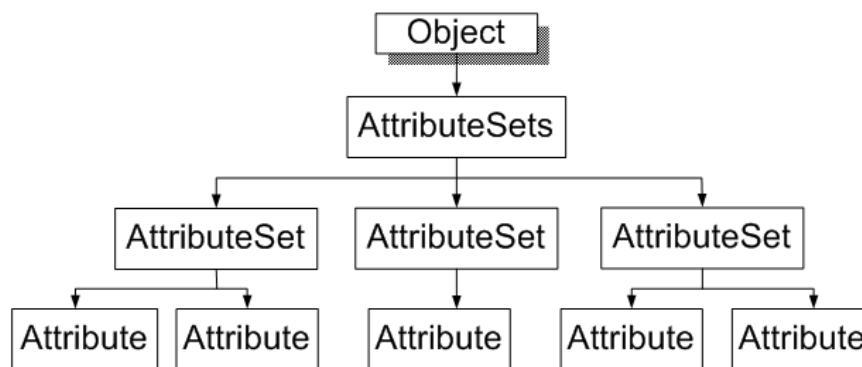


- Attributeは、プログラマがInventorのほとんどのオブジェクト(Document含む)に対して、非グラフィックデータを付加することを可能にする。付加できるデータは、名前付きのデータまたは名前と値のペアとなる
- 同等の機能はGUIには存在しない
- *AttributeManager*は、アトリビュート情報に基づいて、エンティティの問い合わせのメソッドを提供



# アセンブリ

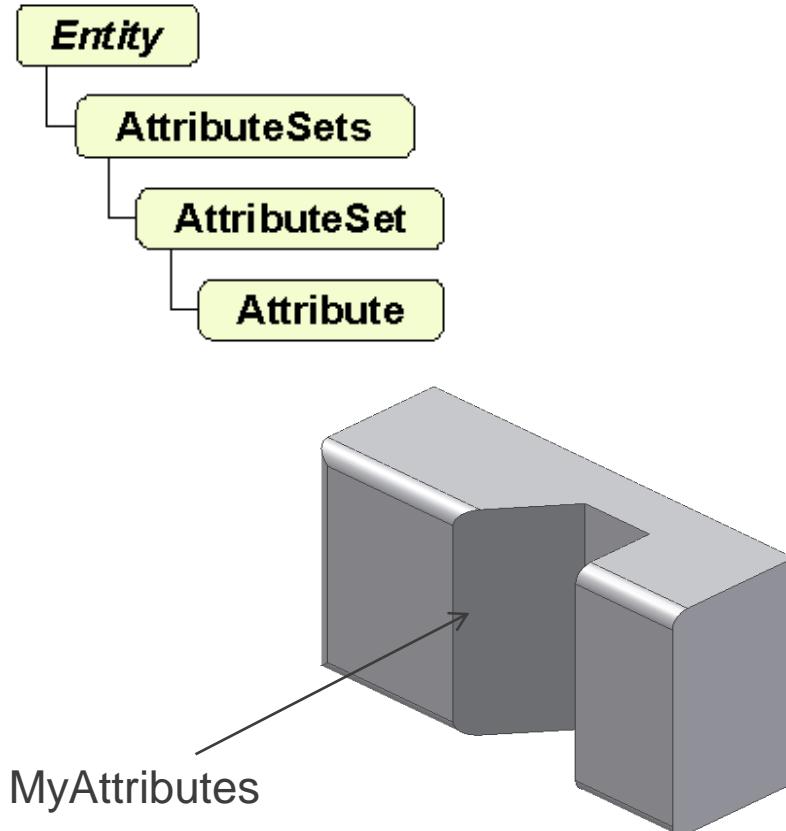
## アтриビュートの利用方法



- アтриビュートの一般的な利用方法は、後からエンティティを見つける為に、アтриビュートを付加する(エンティティに名前を付ける事など)こと
- アтриビュートにより、エンティティに情報を付加し、後から問い合わせることが可能
- カスタムアプリケーションで使用する特定のデータを付加することも可能

# アセンブリ

## アトリビュートの作成



Load = 400  
DirectionX = 0  
DirectionY = 0  
DirectionZ = 1

- アトリビュートは、*AttributeSets*プロパティを持つ全てのオブジェクトでサポート
- それぞれのオブジェクトは、複数のアトリビュートセットを持つことが可能
- アトリビュートセット名は、エンティティ上で一意である必要がある
- それぞれのアトリビュートセットは、複数のアトリビュートを持つことが可能
- アトリビュート名は、アトリビュートセット内で一意である必要がある
- アトリビュート値は、実数、整数、文字、Byte 配列である事が可能

# アセンブリ

## アトリビュートセットとアトリビュートの作成

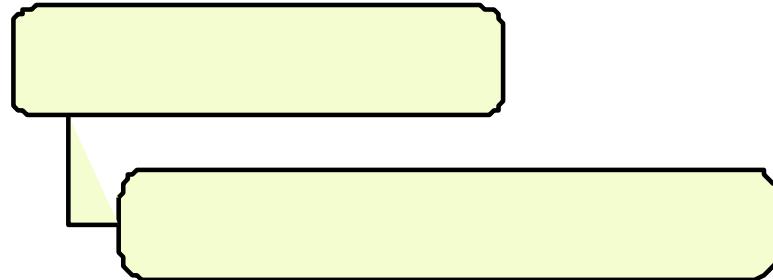
- *AttributeSets* コレクションから、新規*AttributeSet*を追加:
  - ✓ *Public Function Add( ByVal AttributeSetName As String, Optional ByVal CopyWithOwner As Boolean = False ) As AttributeSet*
- *AttributeSet*コレクションから、新規*Attribute*を追加
  - ✓ *Public Function Add( ByVal AttributeName As String, ByVal ValueType As ValueTypeEnum, ByVal Value As Variant ) As Attribute*

*ValueTypeEnum:*

  - *kIntegerType*
  - *kDoubleType*
  - *kStringType*
  - *kByteArrayType*
  - *kBooleanType*

# アセンブリ

## アトリビュートの照会



- *AttributeManager*は、アトリビュートを問い合わせる異なるメソッドをサポート
- アトリビュートセット名やアトリビュート名やアトリビュート値に基づいて、問い合わせすることが可能
  - ✓ *FindAttributes*
  - ✓ *FindAttributeSets*
  - ✓ *FindObjects*
- *Public Function FindObjects( Optional ByVal AttributeSetName As String = "\*", Optional ByVal AttributeName As String = "\*", Optional ByValAttributeValue As Variant ) As ObjectCollection*
- *OpenAttributeSets*は、多くのエンティティにアトリビュートを追加する場合に最適化されたメソッドを提供

# アセンブリ

## アтриビュートの照会のパフォーマンス

- Inventorは、初回の参照時にオブジェクトとAttributeSetのキャッシュを作成する。このキャッシュにより2回目以降のアクセス時間を大幅に短縮が可能
- 引数なしで*FindObject*sメソッドを呼び出すことにより、Inventorに全オブジェクトのキャッシュを作成させることができが可能となり、以後の照会ではキャッシュを使用することができる
- 多くの属性に対して処理を行う場合は、アтриビュートの編集処理をトランザクションまたは、ChangeProcessorを明示的に使用することにより、Inventorに自動トランザクションやUndoメカニズムを使用せず、処理パフォーマンスの改善ができる

# アセンブリ

## アトリビュートのページ

- アトリビュートは、関連するエンティティが既に存在しない状態となってファイルに残る。これは場合によって、エンティティが元に戻ることがあるため
- *AttributeManager.PurgeAttributeSets* メソッドにより、指定したアトリビュートからエンティティが既に存在していないものを削除することが可能

# アセンブリ

## アトリビュートの作成サンプル(VB.net)

```
Public Sub AddAttribute()

    ' 選択されたエッジを取得。エッジが選択状態にある想定
    Dim oEdge As Edge
    oEdge = _InvApplication.ActiveDocument.SelectSet.Item(1)

    ' エッジにattribute set を追加。もしエッジに名前を付けるのみであれば、
    ' ここまでの処理で、後から照会するのに十分であるため次のステップを飛ばしてもよい
    Dim oAttribSet As AttributeSet
    oAttribSet = oEdge.AttributeSets.Add("BoltEdge")

    ' Setにアトリビュートを追加。エッジに付加したい任意の情報を付加可能
    Call oAttribSet.Add("BoltRadius", ValueTypeEnum.kDoubleType, 0.5)

End Sub
```

# アセンブリ

## アトリビュートの作成サンプル(VB.net)

```
Public Sub QueryAttribute()

    ' アクティブドキュメントから、attribute managerを取得
    Dim oAttribMgr As AttributeManager
    oAttribMgr = _InvApplication.ActiveDocument.AttributeManager

    ' 特定のアトリビュートが付加されたオブジェクトを取得
    Dim oObjs As ObjectCollection
    oObjs = oAttribMgr.FindObjects("BoltEdge", "BoltRadius", 0.5)

    ' 指定した名前のアトリビュートセットを持つオブジェクトを取得
    oObjs = oAttribMgr.FindObjects("BoltEdge")

    ' 指定した名前を持つattribute setsを取得
    Dim oAttribSets As AttributeSetsEnumerator
    oAttribSets = oAttribMgr.FindAttributeSets("BoltEdge")

    ' ワイルドカードを使って attribute setsを取得
    oAttribSets = oAttribMgr.FindAttributeSets("Bolt*")

End Sub
```

# アセンブリ

## アтриビュート その他機能

- *AttributeSets*オブジェクトは、XML形式でアтриビュートを出力するDataIOオブジェクトもサポート
- *Attribute*と*AttributeSets*は *Transient*プロパティをサポート。このプロパティにTrueが設定された場合や、*AttributeSet*が*AttributeSets*の*AddTransient*メソッドによって作成された場合、アトリビュートのデータは永続化されません。この機能は、一時的なデータ操作を行いたいが、ドキュメントに余分なデータを付加したくない場合などに有用

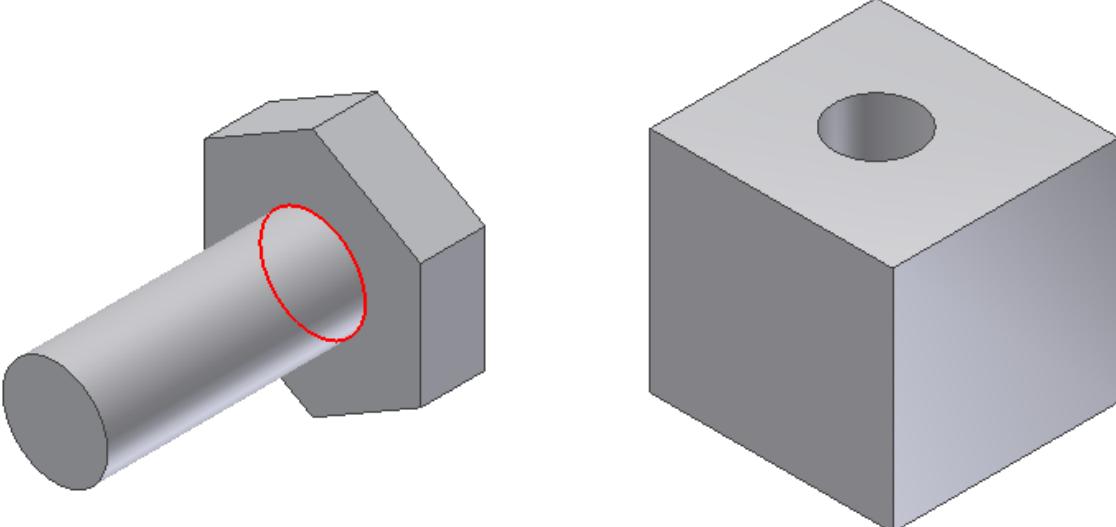
# 演習9

## アтриビュートを利用して拘束を持つアセンブリファイルの作成

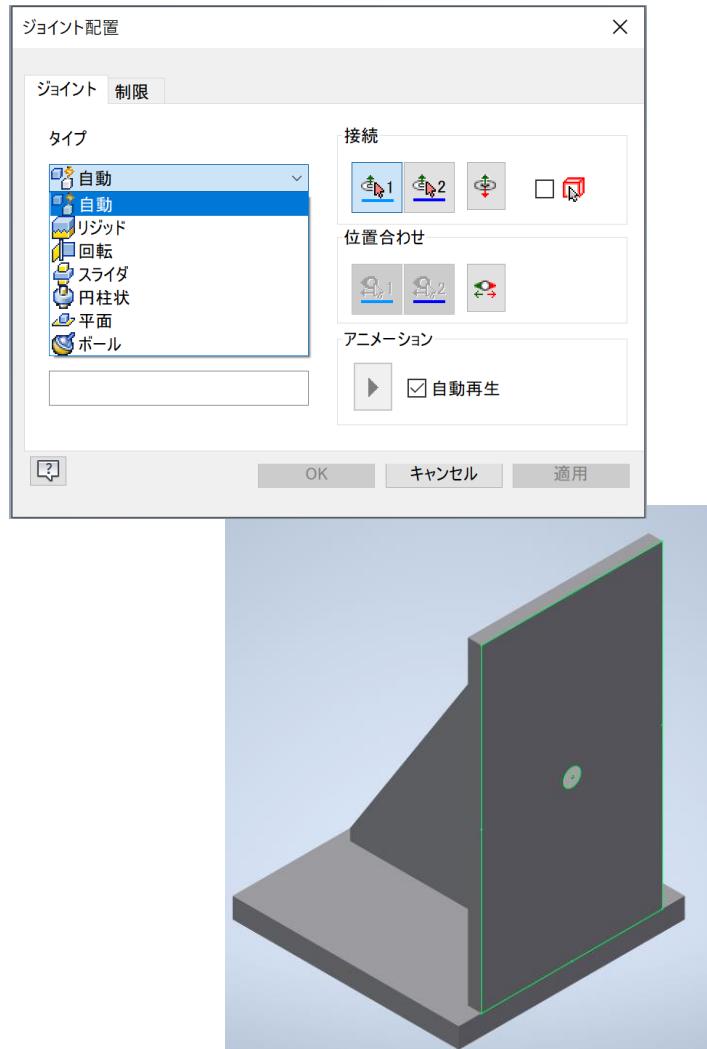
1. 右図に見えるシンプルなボルト状のパーツを作成(Bolt. ipt)
2. シリンダーの面に、アтриビュートを追加するプログラムを作成
3. 右図のような1つの穴を持っているブロックの別のパーツを作成(Block. ipt)
4. ボルトの場合と同様に、ホールにアトリビュートを追加するプログラムを作成
5. 以下のプログラムを作成
  - ✓ アセンブリ内にボルトとブロックを挿入
  - ✓ アトリビュートAPIを使用して、ホールとボルトのフェースを取得
  - ✓ ボルトとホール間に、挿入拘束を作成

### ■ 演習のヒント

- ✓ エッジオブジェクトへのアトリビュートの追加はAttributeSets.Addメソッドを使用します
- ✓ アセンブリへのオカレンスの配置はAssemblyComponentDefinitionオブジェクトのOccurrences.Addメソッドを使用します
- ✓ 插入拘束の作成はAssemblyComponentDefinitionオブジェクトの、Constraints.AddInsertConstraintメソッドを使用します



# アセンブリ ジョイント(Connection)



- コンポーネントの配置とモーションの設定を簡単に定義
- Inventorでは、ボディの関係を表すGUIを表示
- Inventor 2014からAPIによるフルサポート

# アセンブリ

## ジョイント(Connection) APIとサンプルコード(VB.Net)

```
'ジョイントのためのジオメトリ定義のため2つのintentを作成する。  
'2つのフェースがあると想定  
Dim intentOne As GeometryIntent  
intentOne = asmDef.CreateGeometryIntent(Face1,  
    PointIntentEnum.kPlanarFaceCenterPointIntent)  
Dim intentTwo As GeometryIntent  
intentTwo = asmDef.CreateGeometryIntent(Face2,  
    PointIntentEnum.kPlanarFaceCenterPointIntent)  
  
'二つのパート間に、rigidジョイントを作成  
Dim connectDef As AssemblyConnectionDefinition  
connectDef =  
asmDef.Connections.CreateAssemblyConnectionDefinition(  
    AssemblyConnectionTypeEnum.kRigidConnectionType,  
    intentOne, intentTwo)  
connectDef.FlipAlignmentDirection = False  
connectDef.FlipOriginDirection = True  
Dim connect As AssemblyConnection  
connect = asmDef.Connections.Add(connectDef)  
'ジョイントを可視化  
connect.Visible = True
```

- *Connections*

- ✓ アセンブリ内のジョイントのコレクション

- *Connections.CreateAssemblyConnectionDefinition*

- ✓ ジョイント定義を作成

- *AssemblyConnection*

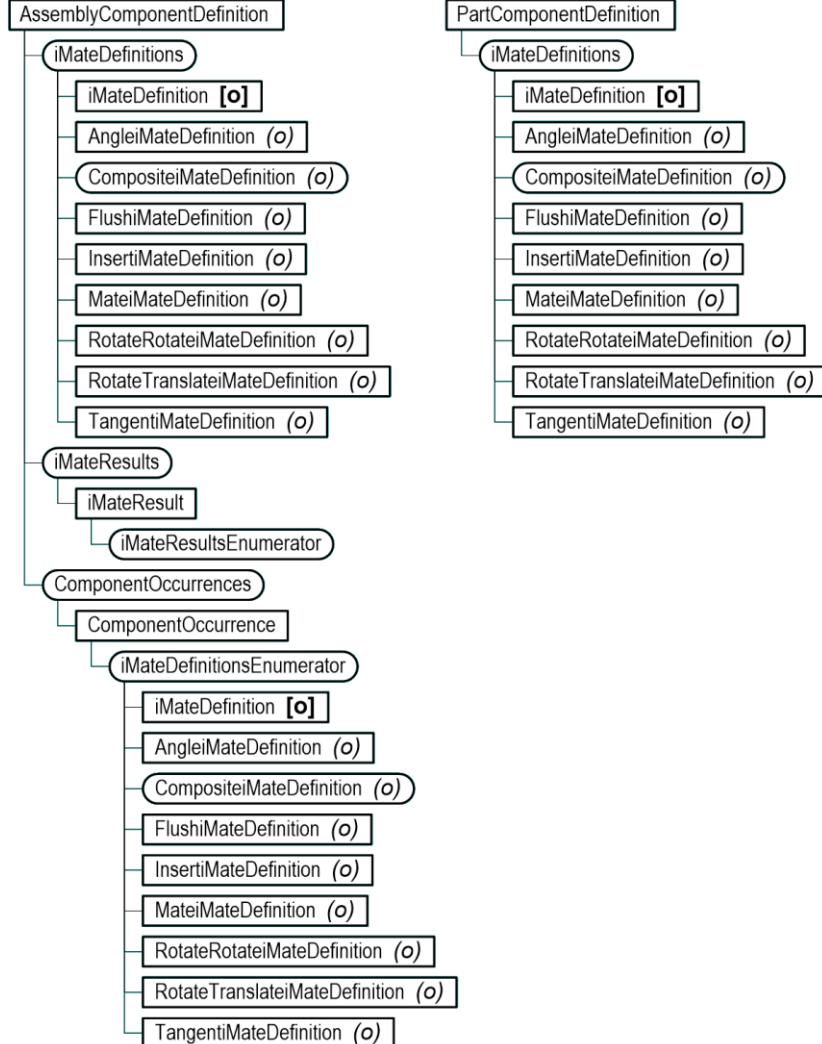
- ✓ ジョイントオブジェクト

- *Connections.Add*

- ✓ 定義をもとに、ジョイントを追加

# アセンブリ

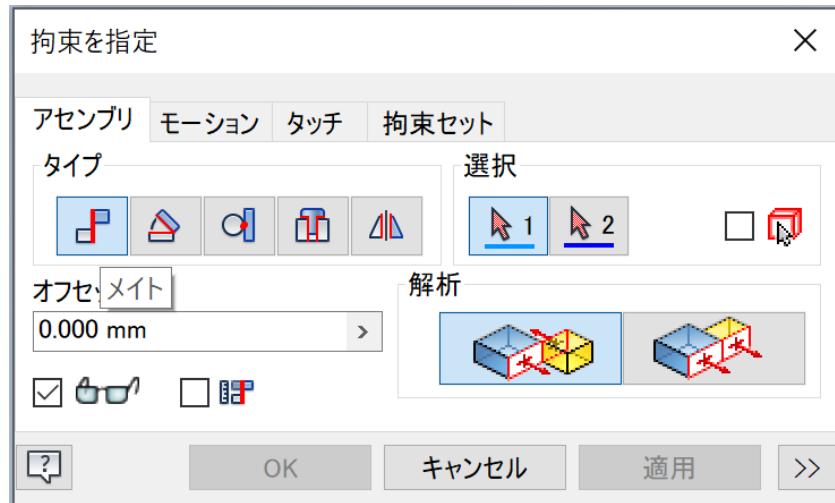
## iMate



- APIではiMateは2つのパートから構成される
  - ✓ iMate 定義… iMateのパートまたはサブアセンブリ
  - ✓ iMate 結果… アセンブリ内の2つのiMate結合の結果

# アセンブリ

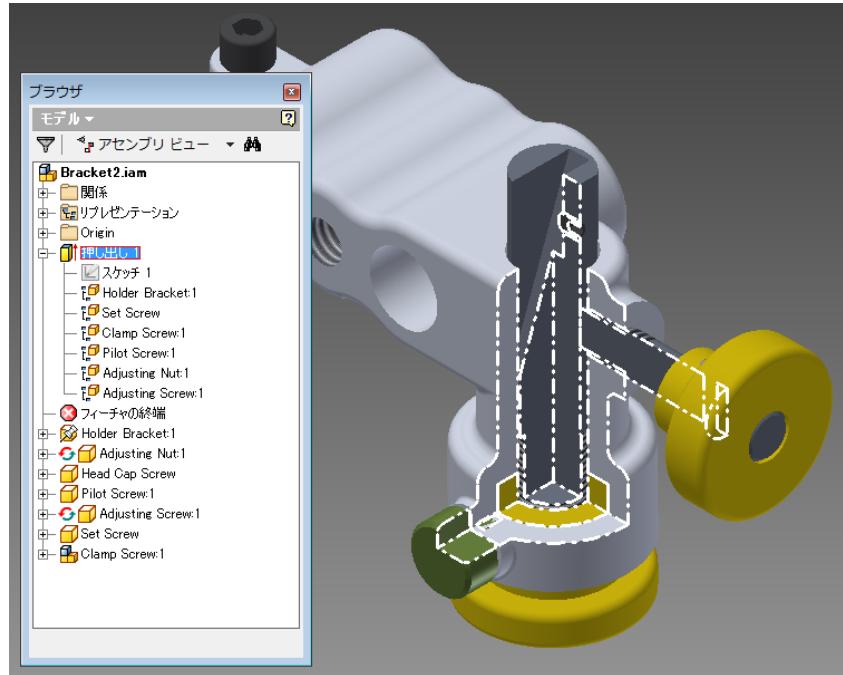
## iMate定義の作成



- *ComponentDefinition.iMateDefinitions*オブジェクトは、様々な追加メソッドを提供
  - ✓ *AddAngleiMateDefinition*
  - ✓ *AddCompositeiMateDefinition*
  - ✓ *AddFlushiMateDefinition*
  - ✓ *AddInsertiMateDefinition*
  - ✓ *AddMateiMateDefinition*
  - ✓ *AddRotateRotateiMateDefinition*
  - ✓ *AddRotateTranslateiMateDefinition*
  - ✓ *AddTangentiMateDefinition*
- 各Addメソッドは、*MatchList*をオプション引数に取る
  - ✓ String配列により、iMate定義の優先順位を指定

# アセンブリ

## アセンブリフィーチャ



- アセンブリフィーチャーはアセンブリにのみ存在し、アセンブリ内のパーツに影響する。元のパーツは変わらない
- API では、パートドキュメントの *PartFeature* オブジェクトを再利用するが、限定的な機能を提供
  - マテリアルを取り除くことのみが可能
  - 部分的な Extent タイプをサポート
  - 全てのフィーチャはサポートされない
- *RemoveParticipant* と *AddParticipant* メソッドは、フィーチャーに含むパーツの制御使用
- *iProperties* は、component 定義からアクセスしたドキュメントを通してアクセス可能

# アセンブリ

## アセンブリフィーチャ作成サンプル(VB.Net)

```
Public Sub AssemblyFeature()
    Dim oAsmDef As AssemblyComponentDefinition
    oAsmDef = _InvApplication.ActiveDocument.ComponentDefinition
    'XY作業平面にスケッチを作成
    Dim oSketch As PlanarSketch
    oSketch = oAsmDef.Sketches.Add(oAsmDef.WorkPlanes.Item(3))
    Dim oTG As TransientGeometry
    oTG = _InvApplication.TransientGeometry
    '矩形を作成
    oSketch.SketchLines.AddAsTwoPointRectangle(oTG.CreatePoint2d(0.1, 0.1), oTG.CreatePoint2d(1, 0.5))
    'プロファイルを作成
    Dim oProfile As Profile
    oProfile = oSketch.Profiles.AddForSolid
    '押し出しを作成
    Dim oExtrudeDef As ExtrudeDefinition
    oExtrudeDef = oAsmDef.Features.ExtrudeFeatures.CreateExtrudeDefinition(oProfile, PartFeatureOperationEnum.kCutOperation)
    oExtrudeDef.SetDistanceExtent("2 cm", PartFeatureExtentDirectionEnum.kSymmetricExtentDirection)

    oAsmDef.Features.ExtrudeFeatures.Add(oExtrudeDef)
End Sub
```

# アセンブリ

## BOM API

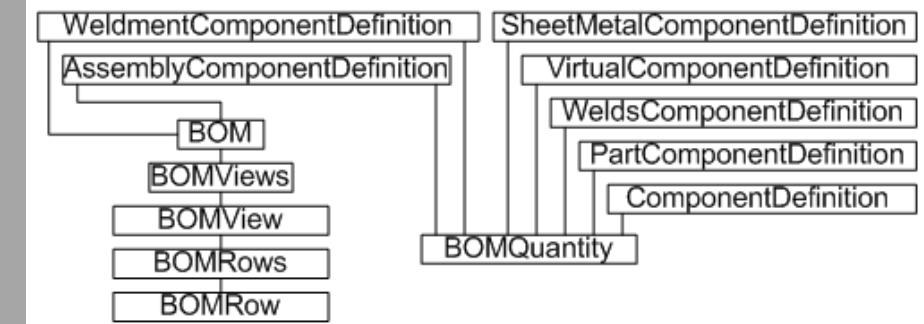
- BOM APIにより、BOMデータの変更や問い合わせ、出力をすることが可能。Inventorは、コンポーネントのBOMデータの保持にPropertySetオブジェクトを利用。

```
Public Sub BomAccess()
    Dim oAsm As AssemblyDocument
    oAsm = _InvApplication.ActiveDocument
    Dim oBOM As BOM
    oBOM = oAsm.ComponentDefinition.BOM
    oBOM.StructuredViewEnabled = True

    Dim oBomView As BOMView
    oBomView = oBOM.BOMViews("Structured")

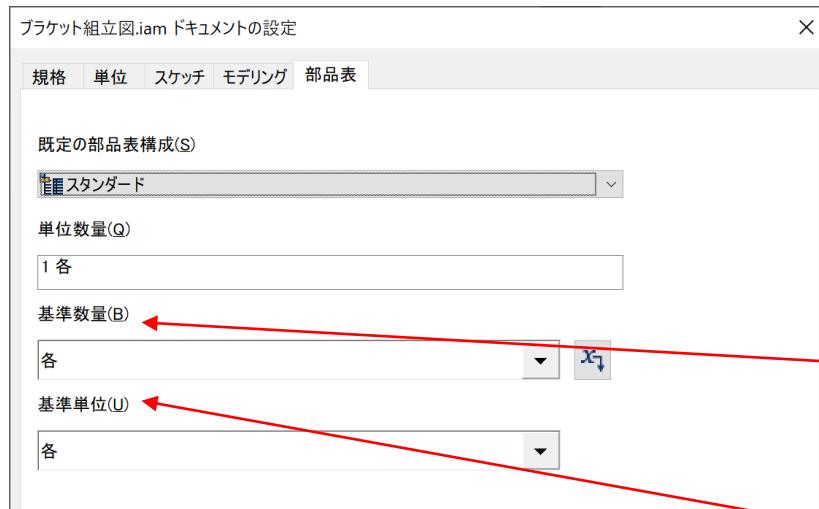
    Dim rowIdx As Long
    For rowIdx = 1 To oBomView.BOMRows.Count

        Dim oRow As BOMRow
        oRow = oBomView.BOMRows(rowIdx)
        Debug.Print("ItemNumber: " & oRow.ItemNumber & " TotalQuantity = " & oRow.TotalQuantity)
        Dim oCompDef As ComponentDefinition
        oCompDef = oRow.ComponentDefinitions(1)
        Dim oDesignPropSet As PropertySet
        oDesignPropSet = oCompDef.Document.PropertySets("Design Tracking Properties")
        Next
    End Sub
```



# アセンブリ

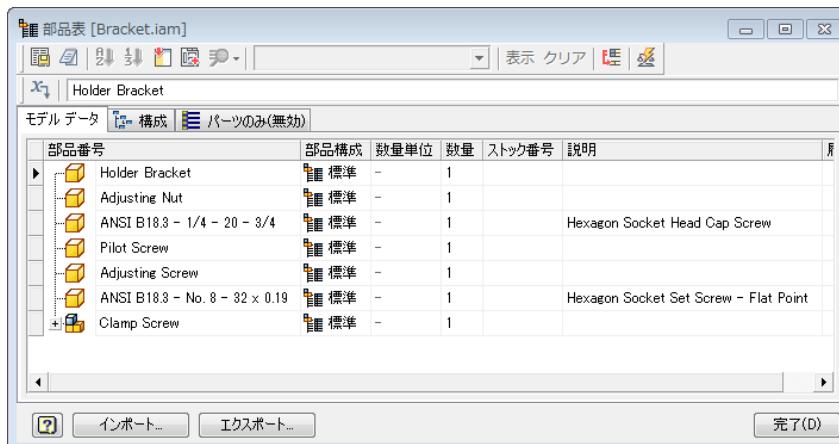
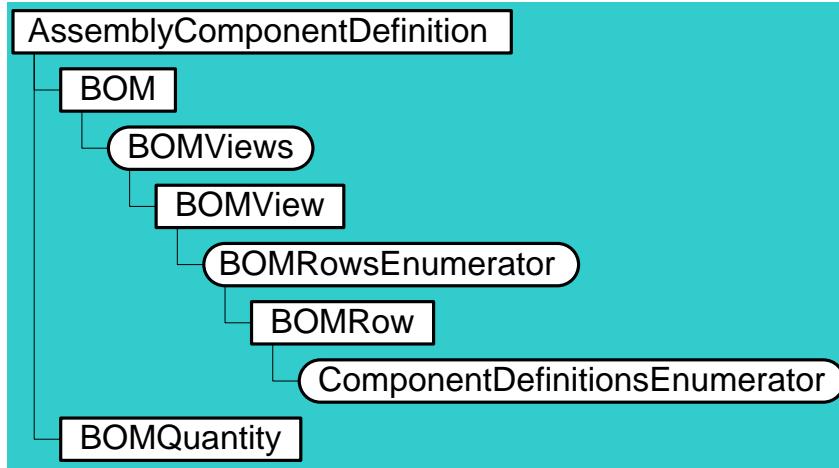
## BOMQuantityオブジェクト



- *BOMQuantity*オブジェクトは、BOM行の単位数量にアクセスするメソッド・プロパティを提供
- ドキュメントの設定ダイアログの部品表タブの設定へのプログラムからのアクセスを提供
- *BOMQuantity.SetBaseQuantity* メソッドは、基準数量の設定を提供
- *BOMQuantity.BaseUnits* プロパティはコンポーネントの基準単位を表す
- *BOMQuantity.UnitQuantity* プロパティは  $BaseQuantity * BaseUnits$  の計算結果を返す

# アセンブリ

## Global Bill of Materials



- 単一のBOMはアセンブリまたは図面から使用される
- パーツリストや図面のバルーンで使用されるアイテムナンバーはBOMでコントロールされる
- `BOMRow.TotalQuantity`は上書きが可能。
- `BOMRow.TotalQuantityOverridden` はプロパティが上書きされているかの確認に使用
- `BOMRow.TotalQuantityOverridden` プロパティには`False`のみを設定可能で、その場合上書きされた値は削除される

# アセンブリ

## Global Bill of Materials



- APIでは、バーチャルコンポーネントの作成と問い合わせをサポート
- バーチャルパーツはアセンブリ内の標準的な *ComponentOccurrence* オブジェクトで表現される
- バーチャルパーツに関連している *ComponentDefinition* は、*VirtualComponentDefinition* オブジェクトとなる

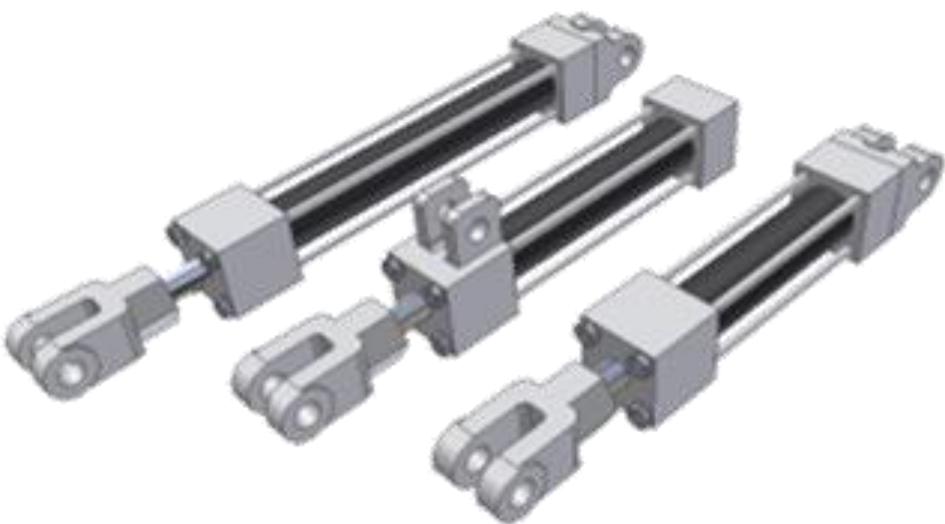
# アセンブリ

## BOMのエクスポート サンプル(VB.net)

```
Public Sub ExportBOM()
    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument
    Dim oBOM As BOM
    oBOM = oDoc.ComponentDefinition.BOM
    '全階層を出力するよう設定
    oBOM.StructuredViewFirstLevelOnly = False
    'StructuredViewを有効化
    oBOM.StructuredViewEnabled = True
    "Structured" BOMViewのリファレンスを取得
    Dim oStructuredBOMView As BOMView
    oStructuredBOMView = oBOM.BOMViews.item("Structured")
    'BOMをエクセルにエクスポート
    oStructuredBOMView.Export("C:\Temp\BOM-StructuredAllLevels.xls", FileFormatEnum.kMicrosoftExcelFormat)
End Sub
```

# アセンブリ

## iAssembly



- APIサポート

- ✓ iAssembly ファクトリの作成
- ✓ 関連する Excel ワークシートへのアクセス
- ✓ ファクトリの行と列カラムへのアクセス
- ✓ アセンブリ内の iAssembly オカレンスの配置 再配置
- ✓ 作図ビューの為に使用するメンバーの指定

# アセンブリ

## iAssemblyオブジェクト

- iAssemblyは、Factoryおよびメンバオブジェクトの存在がiPartsに類似
- *AssemblyComponentDefinition.CreateFactory*はアセンブリをiAssemblyFactoryへ変換。新規に作成された *iAssemblyFactory*オブジェクトは空のエクセルスプレッドシートを持つ
- *AssemblyComponentDefinition.iAssemblyFactory* プロパティはエクセルシートとメンバの作成をする*iAssemblyTable*のアクセスを提供
  - *DefaultRow* プロパティ
  - *TableColumns* プロパティ
  - *TableRows* プロパティ
  - *ExcelWorkSheet* プロパティ
  - *CreateMember* メソッド
- *AssemblyComponentDefinition.iAssemblyMember*は、iAssemblyMemberオブジェクトへのアクセスを提供

# アセンブリ

## iAssembly Memberの追加 サンプル(VB.net)

```
Public Sub AddiAssemblyOccurrence()
    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument

    Dim oOccurrences As ComponentOccurrences
    oOccurrences = oDoc.ComponentDefinition.Occurrences

    Dim oTG As TransientGeometry
    oTG = _InvApplication.TransientGeometry

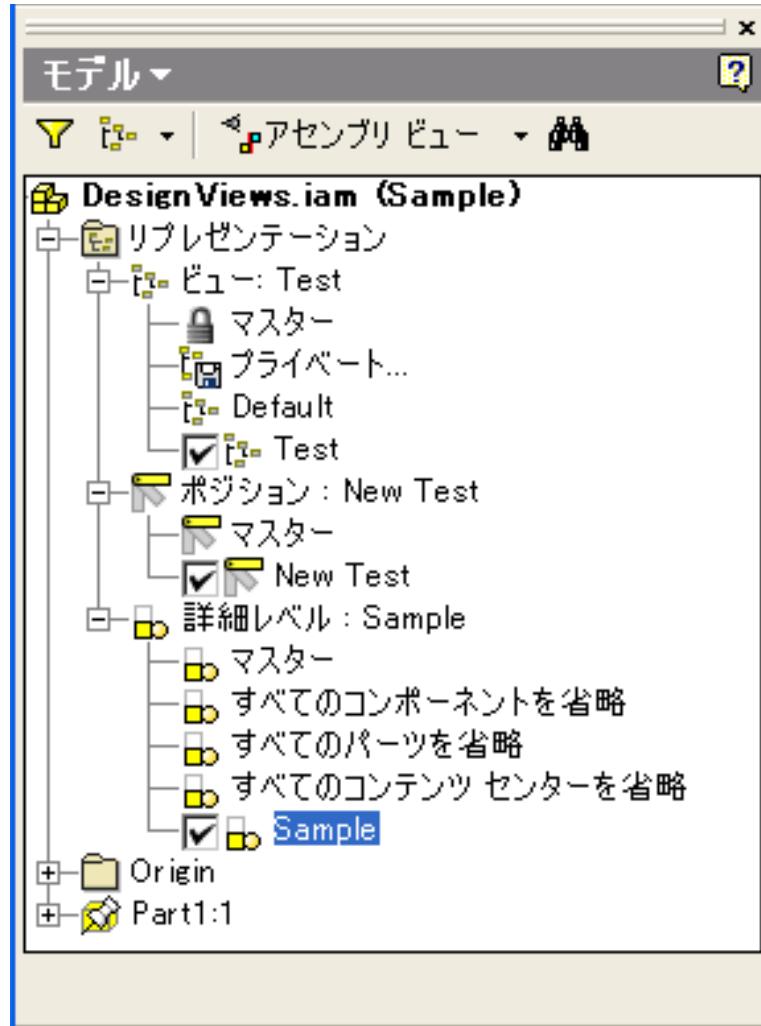
    Dim oPos As Matrix
    oPos = oTG.CreateMatrix
    Dim oNewOcc As ComponentOccurrence

    '行は Long 値 (行インデックス) または、文字列 (メンバー名) で指定
    oNewOcc = oOccurrences.AddiAssemblyMember("C:\MyiAsm.iam", oPos, 1)
    oNewOcc = oOccurrences.AddiAssemblyMember("C:\MyiAsm.iam", oPos, "MemberName")

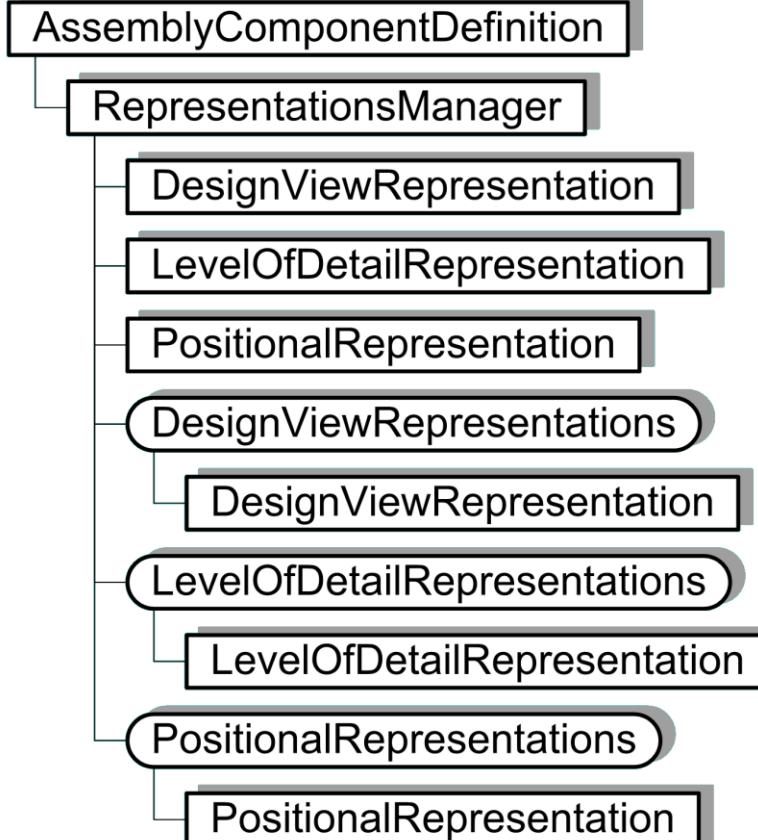
End Sub
```

# アセンブリ

## アセンブリリプレゼンテーション



- デザインビュー・ポジションビュー・詳細レベルを含む



# アセンブリ

## アセンブリリプレゼンテーション サンプル(VB.net)

```
Public Sub DesignViewSample()

    Dim oAsmDef As AssemblyComponentDefinition
    oAsmDef = _InvApplication.ActiveDocument.ComponentDefinition

    Dim oDesignViewReps As DesignViewRepresentations
    oDesignViewReps = oAsmDef.RepresentationsManager.DesignViewRepresentations

    ' ComponentOccurrence 機能を使って状態のセット (visibility, color) など。
    ' デザインビューが作成時、現在のアセンブリ状態をキャプチャする。
    Dim oDesignViewRep As DesignViewRepresentation
    oDesignViewRep = oDesignViewReps.Add("Test")

    ' マスター・デザインビューをアクティブ化する
    oDesignViewReps.Item("Master").Activate()

End Sub
```

# アセンブリ

## ポジションリプレゼンテーション サンプル(VB.net)

```
Public Sub PositionalRepSample()

    Dim oAsmDoc As AssemblyDocument
    oAsmDoc = _InvApplication.ActiveDocument

    Dim oAsmDef As AssemblyComponentDefinition
    oAsmDef = _InvApplication.ActiveDocument.ComponentDefinition

    Dim oPositionalReps As PositionalRepresentations
    oPositionalReps = oAsmDef.RepresentationsManager.PositionalRepresentations

    ' ポジションリプレゼンテーションを新規作成
    Dim oPosRep As PositionalRepresentation
    oPosRep = oPositionalReps.Add("New Test")

    ' 拘束を取得し、値を上書き
    Dim oConstraint As AssemblyConstraint
    oConstraint = oAsmDoc.ComponentDefinition.Constraints.Item(1)
    Call oPosRep.SetConstraintValueOverride(oConstraint, "1 in")

End Sub
```

# アセンブリ

## 詳細レベル(LoD)リプレゼンテーション サンプル(VB.net)

```
Public Sub LevelOfDetail()

    Dim oAsmDef As AssemblyComponentDefinition
    oAsmDef = ThisApplication.ActiveDocument.ComponentDefinition

    Dim oLODReps As LevelOfDetailRepresentations
    oLODReps = oAsmDef.RepresentationsManager.LevelOfDetailRepresentations

    '新規LoDの作成
    Dim oLODRep As LevelOfDetailRepresentation
    oLODRep = oLODReps.Add("My LOD")

    'オカレンスをサプレス
    oAsmDefOccurrences.Item(1).Suppress

    'ドキュメントを保存。LoDも保存される
    ThisApplication.ActiveDocument.Save

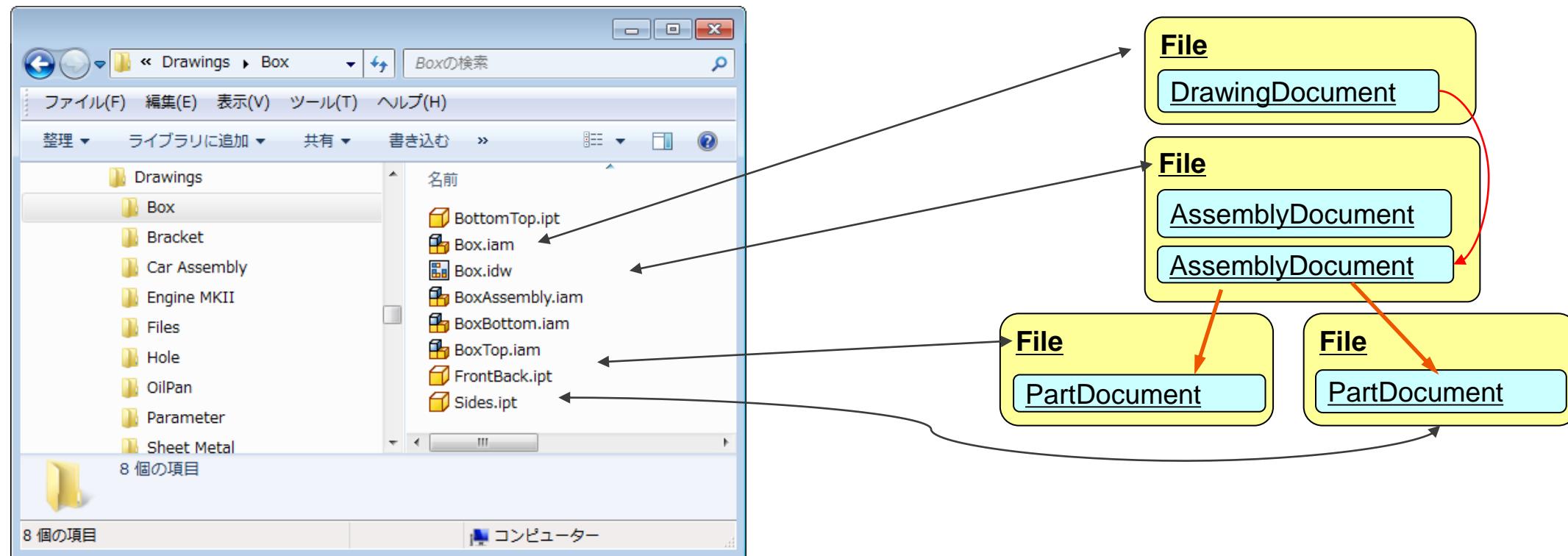
    'マスターLoDをアクティブ化
    oAsmDef.RepresentationsManager.LevelOfDetailRepresentations.Item("Master").Activate

End Sub
```

# アセンブリ

## 詳細レベル (LoD:Level of Detail)

- *File*オブジェクトはディスク上のファイルを表現。ファイルは複数の*Document*オブジェクトを持つことができる
- 各LoDは、*File*内の特定の*Document*に対応
- 全ての*File*は、既定の*Document*を持ち、ほとんどの場合は*Document = File*として扱うことができる



# アセンブリ

## 詳細レベル(LoD) ファイル名

- *Document.FullFilename* プロパティは、そのDocumentを含むFileのファイル名を戻す
- *Document.FullDocumentName* プロパティは、ファイル名とLoD名を連結したドキュメント名を戻す
  - ✓ 例: “C:\Temp\Assembly1.iam<BigParts>”
- 特別なケースはマスターLoD。ファイル名のみを指定した場合は暗黙的にマスターのLoDを指す

*oAsmDoc = oDocuments.Open(“CTemp Assembly1.iam”)*

は以下と同等

*oAsmDoc = oDocuments.Open(“C:Temp Assembly1.iam<Master>”)*

# アセンブリ

## 詳細レベル(LoD)とアセンブリプロパティ

- BOM、MassPropertiesなどの幾つかのプロパティは、マスターLoDから取得する必要がある。マスターLoD以外からは無効なオブジェクトとなる

```
Public Sub BOMfromLoD()

    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument

    Dim oAsmDocMasterLOD As AssemblyDocument
    oAsmDocMasterLOD = _InvApplication.Documents.Open(oDoc.File.FullName, False)

    'BOMをマスターLoDから取得
    Dim oBOM As BOM
    oBOM = oAsmDocMasterLOD.ComponentDefinition.BOM

    '以降取得したBOMに対して操作を行う...
    '以下のコードは、BOM操作のサンプル
    oBOM.StructuredViewFirstLevelOnly = True
    oBOM.StructuredViewEnabled = True

End Sub
```

# アセンブリ

## リファレンスアクセスAPI

- Document
  - *AllReferencedDocuments As DocumentsEnumerator*
  - *ReferencedDocumentDescriptors As DocumentDescriptorsEnumerator*
  - *ReferencedDocuments As DocumentsEnumerator*
  - *ReferencedOLEFileDescriptors As ReferencedOLEFileDescriptors*
  - *ReferencingDocuments As DocumentsEnumerator*
- File
  - *AllReferencedFiles As FilesEnumerator*
  - *AvailableDocuments As DocumentsEnumerator (Inventor) および ApprenticeServerDocuments (Apprentice)*
  - *ReferencedFileDescriptors As FileDescriptorsEnumerator*
  - *ReferencedFiles As FilesEnumerator*
  - *ReferencingFiles As FilesEnumerator*

# アセンブリ

## リファレンスの置き換え

- 新しいリファレンスファイルは置き換え元と同じ資産を持っている必要がある。通常、元ファイルの修正されたコピー
- InventorまたはApprenticeのAPIで直接リファレンス情報を編集可能

```
Public Sub ReplaceReferences()
    Dim oDoc As AssemblyDocument
    oDoc = _InvApplication.ActiveDocument
    oDoc.File.ReferencedFileDescriptors(1).ReplaceReference("C:\CopyPart.ipt")
End Sub
```

- オカレンスを置き換えることにより、間接的にリファレンスが修正される

# アセンブリ

## リファレンスの置き換えサンプル 1/2(VB.Net)

- Apprenticeにより、アセンブリファイルとリファレンスファイルを名前を変えて別の場所への保存が容易に可能

```
Dim mApprenticeApp As Inventor.ApprenticeServerComponent = New ApprenticeServerComponent()

Private Sub SaveWithApprentice()
    Dim NewFolder As String = "C:\Temp\
    Dim AsmFullFilename As String = "C:\Program Files\Autodesk\Inventor 2013\Samples\Models\Tube & Pipe\Tank\Tank.iam"
    Dim oApprenticeDoc As Inventor.ApprenticeServerDocument = mApprenticeApp.Open(AsmFullFilename)
    SaveRec(NewFolder, oApprenticeDoc)
    mApprenticeApp.FileSaveAs.ExecuteSaveCopyAs()
    oApprenticeDoc.Close()
End Sub
```

# アセンブリ

## リファレンスの置き換えサンプル 2/2 (VB.Net)

```
Private Sub SaveRec(ByRef NewFolder As String, ByRef oApprenticeDoc As ApprenticeServerDocument)
    Dim NewFullFilename As String = NewFolder + newFileName(oApprenticeDoc.FullFileName)
    Try
        mApprenticeApp.FileSaveAs.AddFileToSave(oApprenticeDoc, NewFullFilename)
        For Each oRefDoc As ApprenticeServerDocument In oApprenticeDoc.ReferencedDocuments
            SaveRec(NewFolder, oRefDoc)
        Next
    Catch
        'Content Center Parts will fail
    End Try
End Sub
```

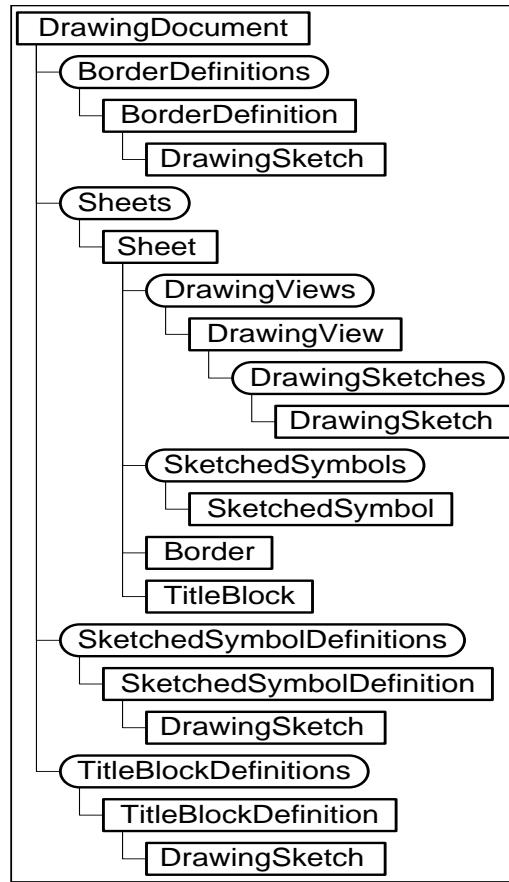
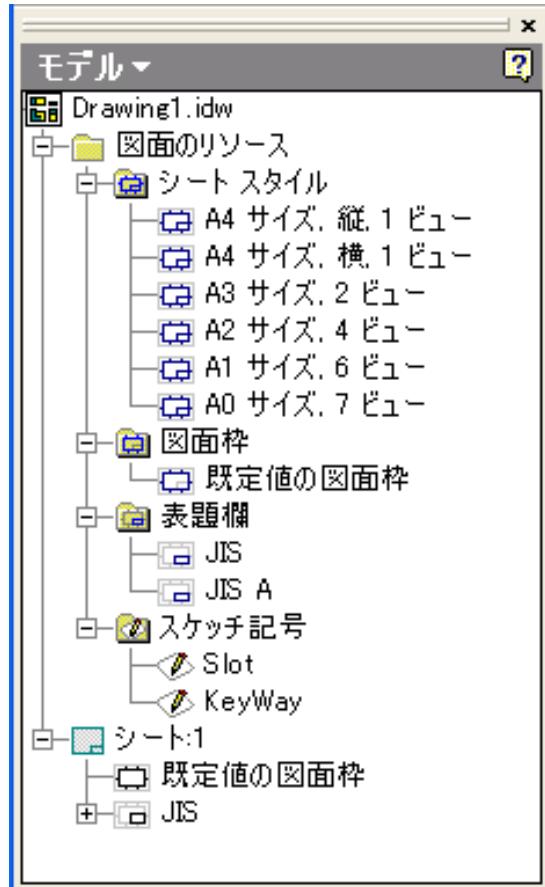
```
Private Function newFileName(ByVal FullFileName As String) As String
    Dim fileInfo As New IO.FileInfo(FullFileName)
    Return fileInfo.Name.Substring(0, fileInfo.Name.Length - 4) + "_new" + fileInfo.Extension
End Function
```



# Drawing ドキュメント

# Drawing ドキュメント

## 図面定義オブジェクト



- BorderDefinitionsコレクション
- TitleBlockDefinitionsコレクション
- SketchedSymbolDefinitionsコレクション

# Drawing ドキュメント

## TitleBlockDefinitionの新規作成サンプル(VB.net)

```
Public Sub AddTitleBlockDef()

    'アクティブドキュメントがDrawingであることを確認
    If Not TypeOf _InvApplication.ActiveDocument Is DrawingDocument Then
        MsgBox("A Drawing Document must be active...")
        Exit Sub
    End If

    Dim oDrawDoc As DrawingDocument
    oDrawDoc = _InvApplication.ActiveDocument

    ' TitleBlockDefinitionsを取得
    Dim oTitleBlks As TitleBlockDefinitions
    oTitleBlks = oDrawDoc.TitleBlockDefinitions

    ' 新規TitleBlockDefinitionを追加
    Dim oTitleBlk As TitleBlockDefinition
    oTitleBlk = oTitleBlks.Add("My TitleBlock")

End Sub
```

# Drawing ドキュメント

## 図面枠(BorderDefinition)の新規作成・編集サンプル(VB.net)

```
Public Sub CreateBorderDefinition()

    Dim oDrawDoc As DrawingDocument
    oDrawDoc = _InvApplication.ActiveDocument

    ' 図面枠を新規作成
    Dim oBorderDef As BorderDefinition
    oBorderDef = oDrawDoc.BorderDefinitions.Add("My Border")

    ' 図面枠のスケッチを編集オープン This is done by calling
    ' BorderDefinitionのEditメソッドで、スケッチを取得
    Dim oSketch As DrawingSketch = Nothing
    Call oBorderDef.Edit(oSketch)

    Dim oTG As TransientGeometry
    oTG = _InvApplication.TransientGeometry

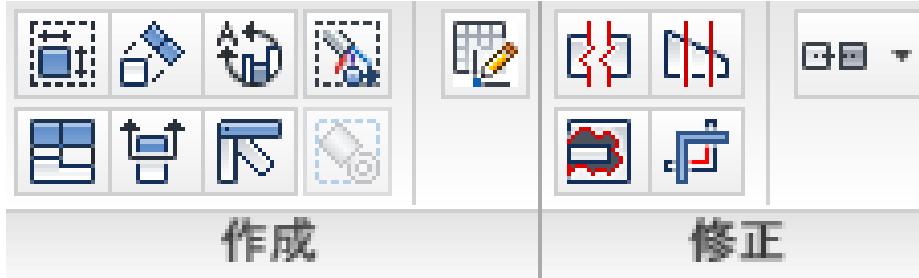
    ' スケッチの機能を使ってジオメトリを追加
    Call oSketch.SketchLines.AddAsTwoPointRectangle(_
        oTG.CreatePoint2d(2, 2), _
        oTG.CreatePoint2d(53.88, 41.18))

    Call oBorderDef.ExitEdit(True)

End Sub
```

# Drawing ドキュメント

## 図面ビュー(DrawingView)の新規作成・編集サンプル(VB.net)



- ベース、投影、断面、詳細、ドラフトビューを作成することが可能
- その他ビュータイプは問い合わせとジェネラルビューとして編集可能
- 図面エンティティのスタイルのコントロール
- 図面ラベルへのフルアクセス
- Suppress/unsuppressにより、ビューの表示をコントロール

# Drawing ドキュメント

## 図面ビュー作成サンプル(VB.net)

```
Public Sub CreateViews()

    Dim oDrawingDoc As DrawingDocument
    oDrawingDoc = _InvApplication.ActiveDocument

    'ビュー作成のためにパートモデルを開く
    Dim oPartDoc As PartDocument
    oPartDoc = _InvApplication.Documents.Open("C:\Temp\Part1.ipt", False)

    Dim oTG As TransientGeometry = _InvApplication.TransientGeometry

    Dim oSheet As Sheet = oDrawingDoc.ActiveSheet

    ' ベースビューを作成
    Dim oFrontView As DrawingView
    oFrontView = oSheet.DrawingViews.AddBaseView(oPartDoc,
                                                oTG.CreatePoint2d(35, 20), 1,
                                                ViewOrientationTypeEnum.kFrontViewOrientation,
                                                DrawingViewStyleEnum.kHiddenLineDrawingViewStyle)

    ' 投影ビューの作成
    Dim oRightView As DrawingView
    oRightView = oSheet.DrawingViews.AddProjectedView(oFrontView,
                                                    oTG.CreatePoint2d(15, 20),
                                                    DrawingViewStyleEnum.kFromBaseDrawingViewStyle)

    Dim oIsoView As DrawingView
    oIsoView = oSheet.DrawingViews.AddProjectedView(oFrontView,
                                                 oTG.CreatePoint2d(15, 35),
                                                 DrawingViewStyleEnum.kHiddenLineRemovedDrawingViewStyle)

End Sub
```

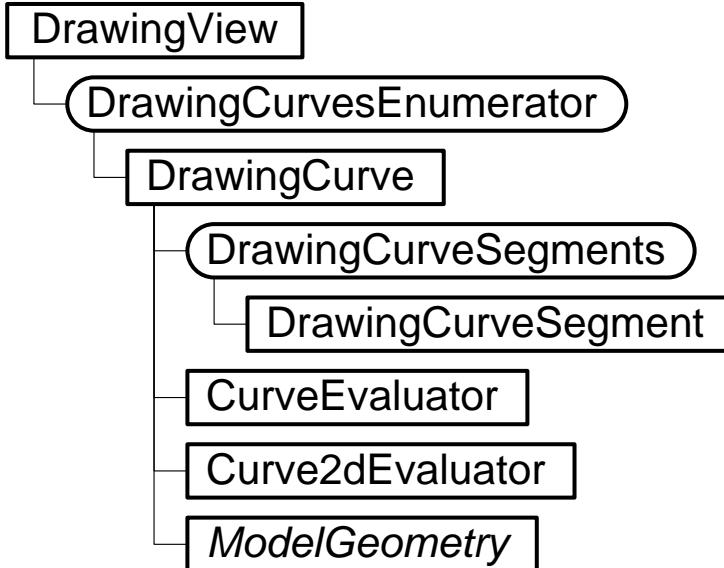
# Drawing ドキュメント

## ドラフトビュー

- ビュータイプはAPIでのみアクセス可能
- 関連ドラフトビューはドラフトビューと標準作図ビューの組み合わせ
  - ✓ 関連ドラフトビューはスケッチのみを持つことができ、どんなモデルジョメトリも表示しない
  - ✓ 関連ドラフトビューはパーツ・アセンブリまたはプレゼンテーションドキュメントの参照を持つ。
- *Sub AddAssociativeDraftView(Model As Document,  
Position As Point2d,  
ByRef Scale As [defaultValue(1.0)] double,  
ByRef Name As [defaultValue("")] BSTR,  
Result As [out, retval] DrawingView\*)*

# Drawing ドキュメント

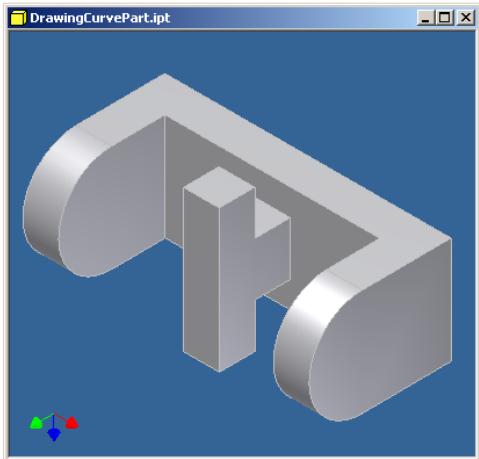
## 図面曲線(Drawing Curve)



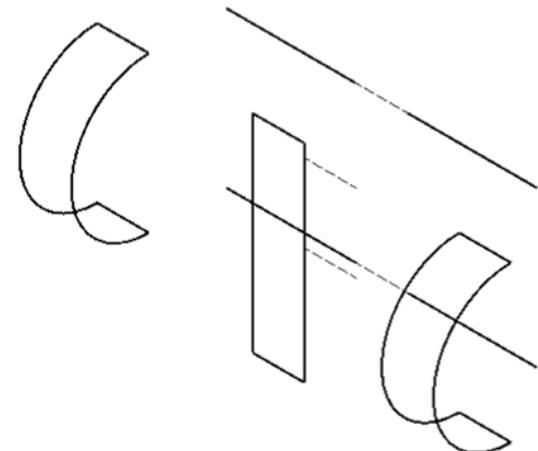
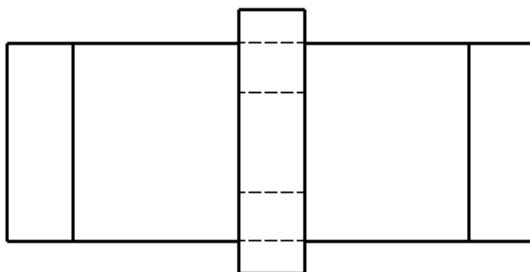
- APIは、パートまたはアセンブリのビューを作成時に作成されたジオメトリへのアクセスを提供
- 図面曲線(Drawing Curve)は、3Dモデルのクリーンアップされたリプレゼンテーションを構築することによって作成される
- 対応するモデルエッジまたはフェースは、曲線から利用可能

# Drawing ドキュメント

## 図面曲線(Drawing Curve)



- ビューで表現されている全てのカーブまたは特定のモデルエッジまたはフェースを表すカーブにアクセス可能。
  - ✓ *DrawingView.DrawingCurves([ModelObject]) As DrawingCurvesEnumerator*
- モデルエッジまたはフェースが作図内に現されないときは何も返さない。
- 選択は、*DrawingCurveSegment*を返す



# Drawing ドキュメント

## DrawingCurveSegment

- 図面曲線の一セグメントを表現
- *GeometryType*プロパティは、ジオメトリオブジェクトのタイプを返す
- *Geometry*プロパティは、シート空間でこの図面の曲線セグメントを表す 2D ジオメトリオブジェクトを返す読み取り専用プロパティ。図面上の実際の曲線は、3D ジオメトリであり、これはフラット化された 2D バージョン。結果としてジオメトリのタイプが異なることがあり、たとえば、円が完全にエッジで表示される場合、3D の円がフラットになり、2D の楕円または線分になる場合がある。この曲線のパラメータ化とオリジナルの 3D 曲線間に予期される相関は無い

# Drawing ドキュメント

## GeometryIntentオブジェクト

- *Sheet.CreateGeometryIntent(Geometry As Object, [Intent]) As GeometryIntent*
  - ✓ *Geometry*…ジオメトリを指定するオブジェクト。有効な入力はDrawingCurve、すべての図面スケッチエンティティ、DrawingDimension、Centerline、Centermark及び Point2dオブジェクト
  - ✓ *Intent*…入力ジオメトリのインテント点を指定するオプション引数。PointIntentEnum の値、ジオメトリ(インテントが 2 つのジオメトリの交差の場合)、ジオメトリ上のシート点を指定する Point2d オブジェクト、入力曲線ジオメトリのパラメータを示す倍精度値(0 から 1)を指定可能。有効なパラメータ値の範囲は、曲線の評価子に GetParamExtents メソッドを使用して取得できる。

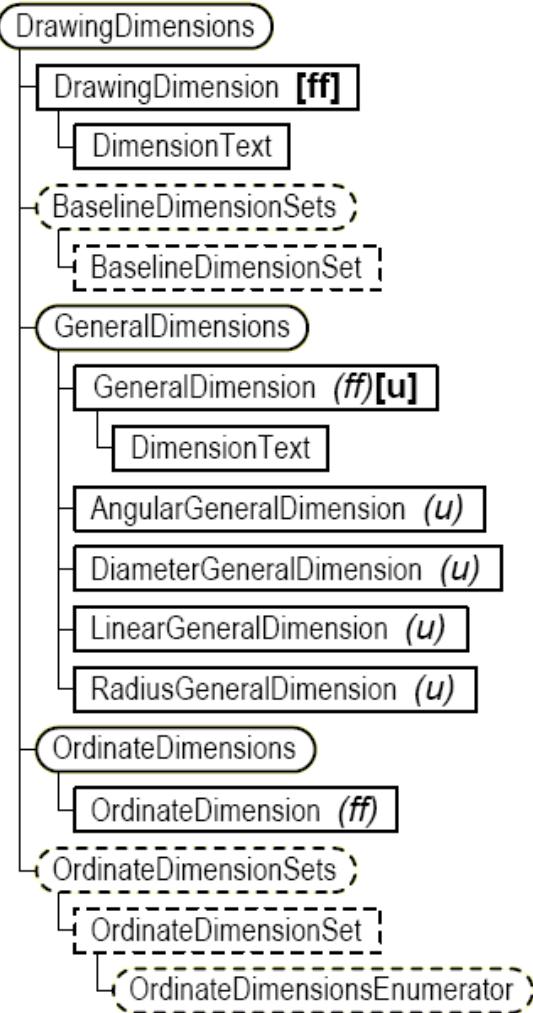
# Drawing ドキュメント

## 入出力としての曲線使用

- 作図曲線(Drawing Curve)は、作図注釈を作成するとき「入力」として提供される
- *DrawingCurve*は、通常カーブに沿って注釈を配置するために必要とされる
- *GeometryIntent*オブジェクトは、作図の注釈オブジェクトとともにに入出力として使用される
- *GeometryIntent*オブジェクトは、作図曲線と曲線上の位置を定義

# Drawing ドキュメント

## 寸法(Dimension)



- DrawingDimension は全ての寸法のベースクラス
- 各寸法に対応するクラスが存在
- 並列寸法、累進寸法、累進寸法セットの作成
- 一般寸法を作成するために、モデルおよびスケッチ寸法を取得可能 (*GeneralDimensions.Retrieve*)



# Drawing ドキュメント

## 寸法(Dimension)の作成サンプル(VB.Net)

```
'モデルから2つのエッジを取得
Dim oEdge1 As Edge
Dim oEdge2 As Edge
oEdge1 = oPartDef.SurfaceBodies.Item(1).Edges.Item(5)
oEdge2 = oPartDef.SurfaceBodies.Item(1).Edges.Item(11)

'エッジに関する作図曲線 (drawing を得る
'作図内にエッジが見つからない場合は Nothing を返す
Dim oCurves1 As DrawingCurvesEnumerator
Dim oCurves2 As DrawingCurvesEnumerator
oCurves1 = oView.DrawingCurves(oEdge1)
oCurves2 = oView.DrawingCurves(oEdge2)

' 曲線( Curve )のエンド点の為の GeometryIntent オブジェクトの作成
Dim oIntent1 As GeometryIntent
Dim oIntent2 As GeometryIntent
oIntent1 = osheet.CreateGeometryIntent(oCurves1.Item(1), PointIntentEnum.kEndPointIntent)
oIntent2 = osheet.CreateGeometryIntent(oCurves2.Item(1), PointIntentEnum.kStartPointIntent)

' 寸法作成
Dim oDim As LinearGeneralDimension
oDim = osheet.DrawingDimensions.GeneralDimensions.AddLinear(oTG.CreatePoint2d(8, 16), _
    oIntent1, oIntent2, DimensionTypeEnum.kVerticalDimensionType)

'エンティティと点を指定しない定義の GeometryIntent オブジェクトの作成
Dim oIntent3 As GeometryIntent
oIntent3 = oSheet.CreateGeometryIntent(oCurves2.Item(1))

'シングルエンティティの寸法作成
oDim = osheet.DrawingDimensions.GeneralDimensions.AddLinear(oTG.CreatePoint2d(15, 23), _
    oIntent3, , DimensionTypeEnum.kHorizontalDimensionType)
```

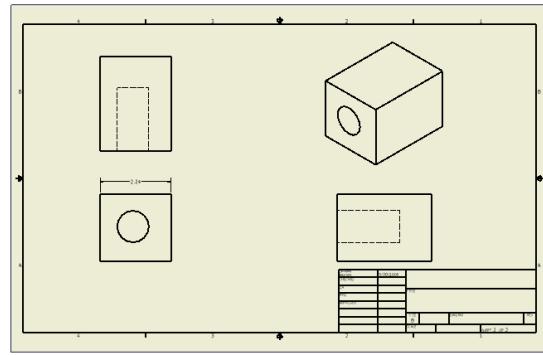
# 演習10

## シートとビューを持つ図面の作成

- 新規 Drawing ドキュメントの作成
- 新規 B サイズシートの作成
- 規定ボーダーの追加
- “JIS” タイトルブロックの追加
- 先回のアセンブリ演習で使ったパートのブロックの正面、右、上面、等角ビューの配置
- 正面ビューのブロックの一番上のエッジに寸法の配置

### ■ 演習のヒント

- ✓ ビューを配置するには、シートオブジェクトの DrawingViews.AddBaseView および DrawingViews.AddProjectedView メソッドを使用します。
- ✓ 直線寸法を作成するにはシートオブジェクトの DrawingDimensions.GeneralDimensions.AddLinear メソッドを使用します。



# Drawing ドキュメント

## Drawing Sketches

- *DrawingSketch* オブジェクトはパートの *PlanarSketch* オブジェクトと同様に図面エンティティと拘束をサポート
- 3次元の位置情報に代わり、図面に固有の変換機能をサポート  
DrawingViewToModelSpace, DrawingViewToModelTransform, DrawingViewToSheetSpace, DrawingViewToSheetTransform, ModelToDrawingViewSpace, ModelToDrawingViewTransform, ModelToSheetSpace, ModelToSheetTransform, SheetToDrawingViewSpace, SheetToDrawingViewTransform, SheetToModelSpace, SheetToModelTransform

# Drawing ドキュメント

## Drawing Sketches

- スケッチはシート、図面ビューおよび様々な定義オブジェクトに関連付けて作成が可能
- 図面スケッチは、編集前にGUIで開いておく必要がある。 参照は、オープンせずに行うことが可能
- スケッチエンティティに、ラインタイプ、スケール、重量および色を設定可能
- 閉じたスケッチに色を塗りつぶしが可能。パートにフィーチャを作成する際に使用するProfileオブジェクトを使用

# Drawing ドキュメント

## Drawing Sketchesサンプル(VB.Net)

```
Public Sub CreateSketch()

    Dim oDoc As DrawingDocument
    oDoc = _InvApplication.ActiveDocument

    Dim oSheet As Sheet
    oSheet = oDoc.ActiveSheet

    ' スケッチを作成
    Dim oSketch As DrawingSketch
    oSketch = oSheet.Sketches.Add

    ' 編集のために、GUIでスケッチを開く
    oSketch.Edit()

    ' 円を追加
    oSketch.SketchCircles.AddByCenterRadius( _
        _InvApplication.TransientGeometry.CreatePoint2d(8, 8), 2)

    ' 編集の終了
    oSketch.ExitEdit()

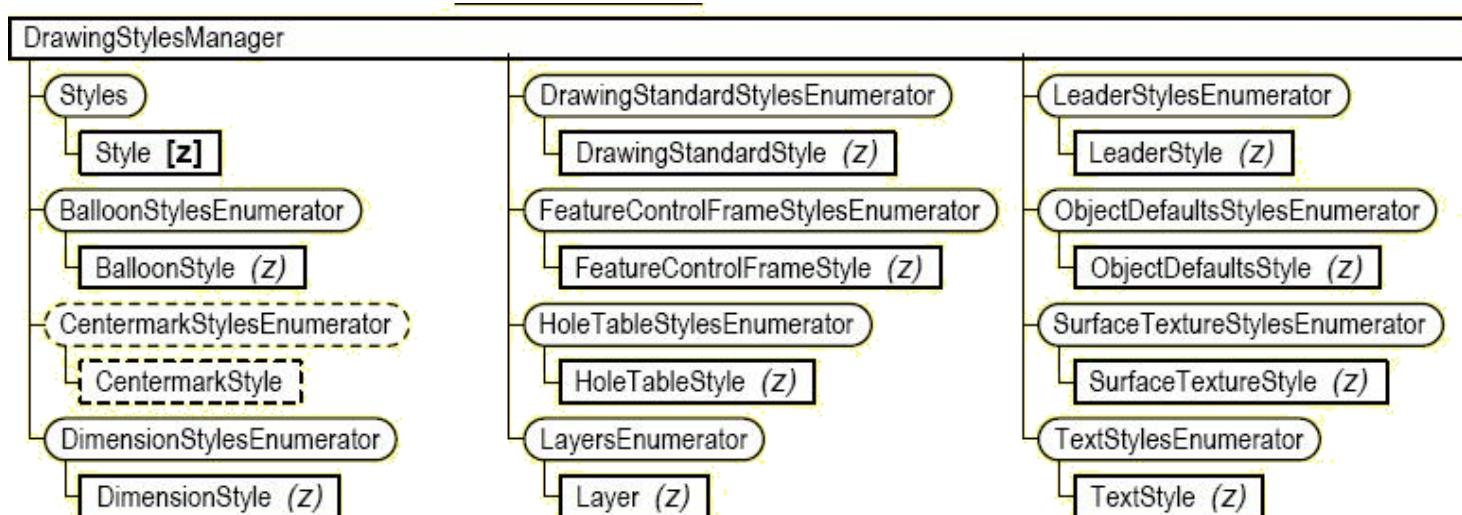
End Sub
```

# Drawing ドキュメント

## 図面Style

- Can create, query, and edit:

- ✓ 尺寸スタイルおよびテキストスタイル
- ✓ フィーチャーコントロールフレームおよびスタイル
- ✓ サーフェーステクスチャシンボルおよびスタイル
- ✓ ホールテーブルおよびスタイル
- ✓ リーダスタイルおよびバルーンスタイル
- ✓ パーツリスト、リビジョン、カスタムテーブルおよびスタイル



# Drawing ドキュメント

## 寸法Style

- *DimensionStyle*オブジェクトは寸法の表示をコントロールするプロパティへのRead/Writeアクセスを提供

DimensionStyle		
Properties		Methods
AlternateDecimalMarkerType	Name	ConvertToLocal
AlternateDisplayUnitType	Parent	Copy
AlternateLeadingZeroDisplay	Prefix	Delete
AlternateLinearPrecision	PrefixAndSuffixOrder	SaveToGlobal
AlternateLinearUnits	Suffix	UpdateFromGlobal
AlternateStyleFormat	TextStyle	
AlternateTrailingZeroDisplay	Tolerance	
AlternateUnitsDisplay	ToleranceAlternateUnitPrecision	
AngularPrecision	ToleranceAngularPrecision	
Application	ToleranceFontSize	
DecimalMarkerType	ToleranceLeadingZeroDisplay	
DisplayUnitType	ToleranceLinearPrecision	
InUse	ToleranceTextStyle	
InternalName	ToleranceTrailingZeroDisplay	
LeadingZeroDisplay	TrailingZeroDisplay	
LinearPrecision	Type	
LinearUnits	UpToDate	
Local		

# Drawing ドキュメント

## 寸法Styleの作成 サンプル(VB.net)

```
Public Sub CreateStyles()

    Dim oDoc As DrawingDocument
    oDoc = _InvApplication.ActiveDocument

    'get Drawing Styles Manager
    Dim oDStylesMan As DrawingStylesManager
    oDStylesMan = oDoc.StylesManager

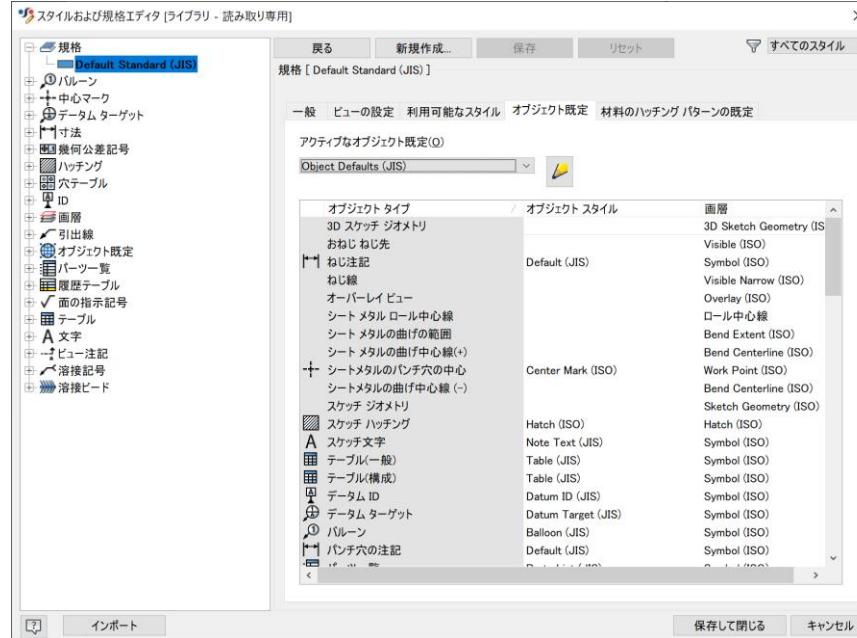
    ' 既存のスタイルをコピーして新規にテキストスタイルを作成
    'the new name is "MyNewTextStyle"
    Dim oNewTextStyle As TextStyle
    oNewTextStyle = oDStylesMan.TextStyles("Label Text (ANSI)").Copy("MyNewTextStyle")

    ' 作成したスタイルのプロパティを変更
    oNewTextStyle.FontSize *= 2
    oNewTextStyle.Italic = True

End Sub
```

# Drawing ドキュメント

## 図面標準とオブジェクトデフォルトスタイル



- *DrawingStandardStyle*
  - ✓ 図面標準を表現
  - ✓ スタイル及びオブジェクトスタイル(*ObjectDefaultsStyle*)の全般的な設定を保持
  - ✓ 各*DrawingStandardStyle*は一つの*ObjectDefaultsStyle*を持つ
- *DrawingStylesManager.ActiveStandardStyle*
  - ✓ 現在アクティブな標準スタイル
  - ✓ 標準をアクティブ化する際に*ObjectDefaultsStyle*も設定

# Drawing ドキュメント

## 図面スタイル作成サンプル(VB.net)

‘以下サンプルコードは前々ページからの継続

‘Drawing Styles Managerを取得

```
Dim oNewDrawingStyle As DrawingStandardStyle  
oNewDrawingStyle = oDStylesMan.StandardStyles(1).Copy("MyNewDSStyle")
```

‘設定を変更

```
oNewDrawingStyle.LinearUnits = UnitsTypeEnum.kCentimeterLengthUnits  
  
Dim oNewObjDefaultStyle As ObjectDefaultsStyle  
oNewObjDefaultStyle = oDStylesMan.ObjectDefaultsStyles(1).Copy("MyNewObjStyles")
```

‘プロパティを変更(図枠のテキストスタイルにMyNewTextStyleを指定)

```
oNewObjDefaultStyle.BorderTextStyle = oNewTextStyle
```

‘新しいオブジェクトスタイルを使うように指定

```
oNewDrawingStyle.ActiveObjectDefaults = oNewObjDefaultStyle
```

‘ドキュメントが新しい DrawingStandardStyleを使うように指定

```
oDStylesMan.ActiveStandardStyle = oNewDrawingStyle
```

# Drawing ドキュメント

## 注釈(Annotative Objects)

- 並列寸法、累進寸法および累進寸法セットの作成
- センターライン、センターマークの自動作成
- スタイルを含むセンターマークおよびセンターラインの追加機能
- 注釈を付加するための寸法及び寸法線へのアクセス
- 曲げ、面取り、穴とねじ及びパンチ穴ノートの作成



# Drawing ドキュメント

## 並列寸法セット

- *BaselineDimensionSets.Add*

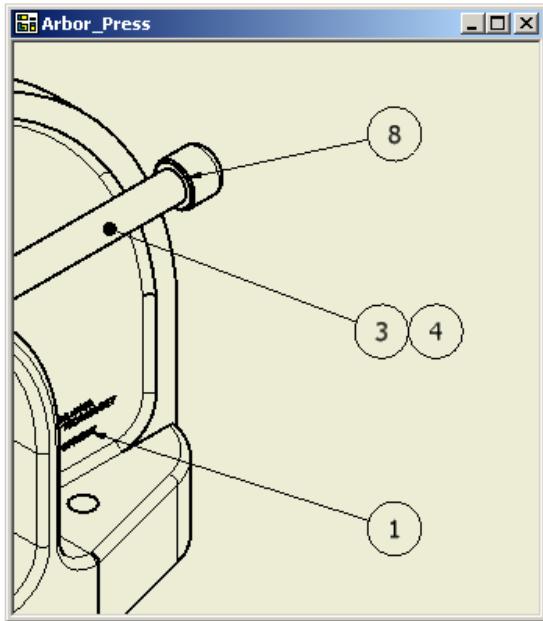
- ✓ GeometryIntents…GeometryIntent オブジェクトを含む ObjectCollection の入力値
- ✓ PlacementPoint…シート上の寸法セットの配置ポイントを指定する入力 Point2d オブジェクト
- ✓ DimensionType…寸法の型を指定する DimensionTypeEnum の入力値

‘垂直並列寸法の作成

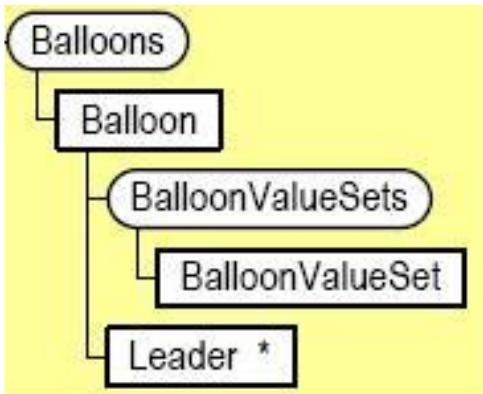
```
Dim oBaselineSet As BaselineDimensionSet  
oBaselineSet = oBaselineSets.Add(oIntentCollection,  
                                 oPlacementPoint,  
                                 DimensionTypeEnum.kHorizontalDimensionType)
```

# Drawing ドキュメント

## バルーン



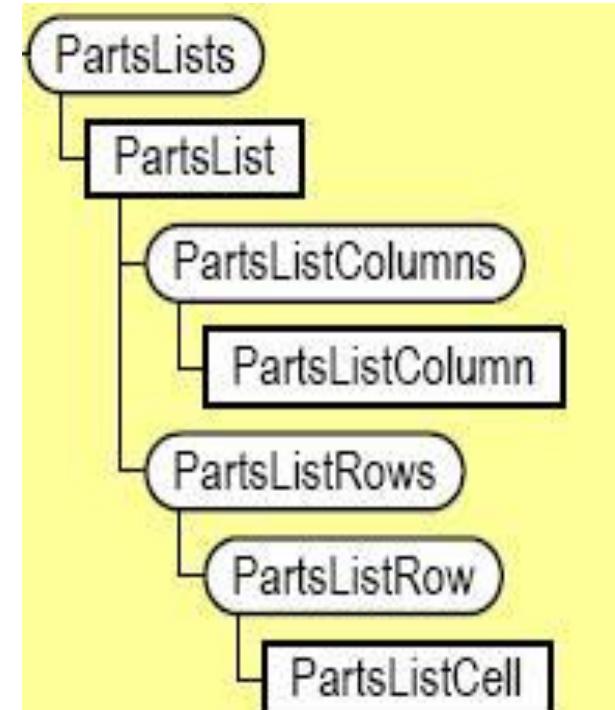
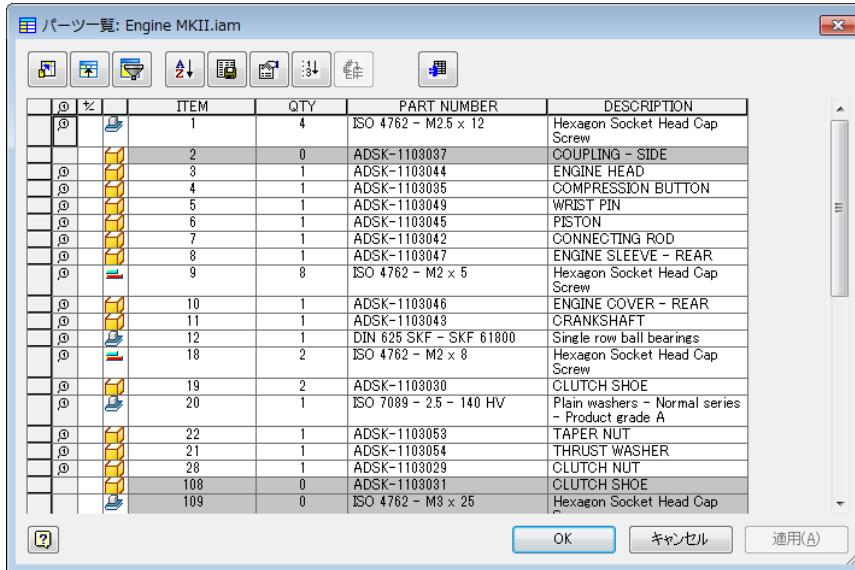
- 新規追加、編集および全ての既存バルーンへのアクセス
  - ✓ バルーン位置と方向
  - ✓ BOMからのバルーン値（アイテム番号）
  - ✓ バルーンテキストの上書き
  - ✓ バルーンタイプ
- 削除機能
- 値を編集することでパーツリスト内のアイテム番号を変更
- バルーンのアイテム番号はBOMで関連するドキュメントを見つけることに使用可能



# Drawing ドキュメント

## パーツリスト

- パーツリストのほとんどの機能をAPIから使用可能



# Drawing ドキュメント

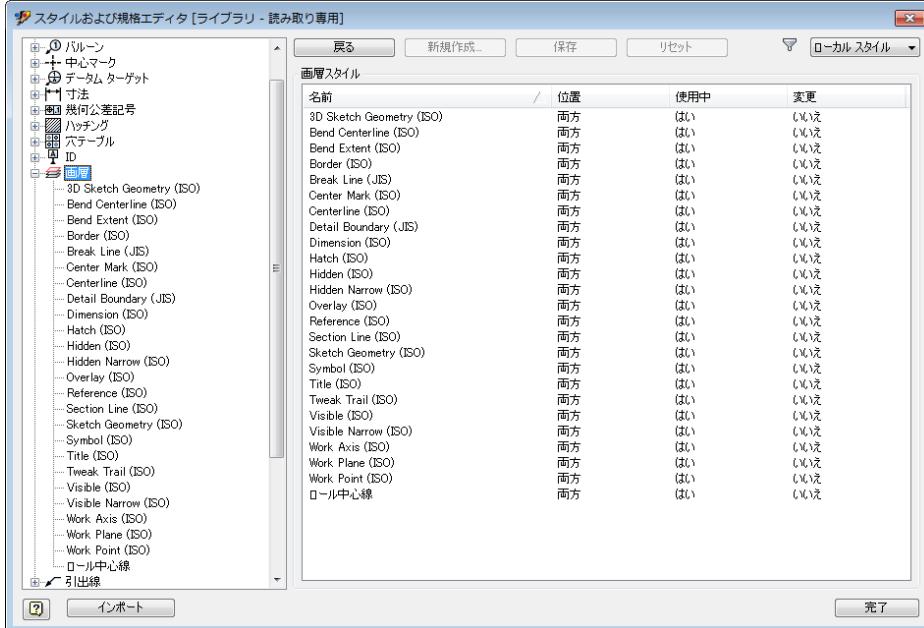
## カスタムテーブル

My Table		
Part Number	Quantity	Material
1	1	Brass
2	2	Aluminium
3	1	Steel

- 作成とテーブル編集のサポート
- エンドユーザーはテーブルサイズ、行と列のサイズをインタラクティブに修正が可能
- エクセルのテーブルをもとにテーブルを作成
- シートメタルの折り曲げテーブルの作成
- コンフィギュレーションテーブル (iPart or iAssembly) の作成

# Drawing ドキュメント

## レイヤー

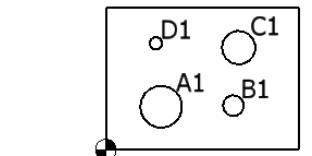
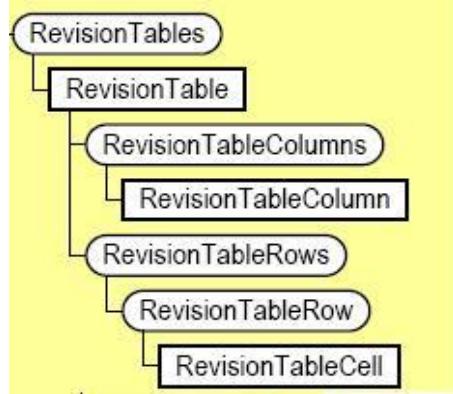


- レイヤーの作成(既存レイヤーのコピーによる)
- 既存レイヤーへのアクセスと編集
- 特定エンティティに関連したレイヤーの設定(APIで現在サポートされている以下のエンティティに限定)
  - ✓ スケッチエンティティ、寸法、バルーン、パーティリスト、履歴ブロック、カスタムテーブル、スケッチシンボル

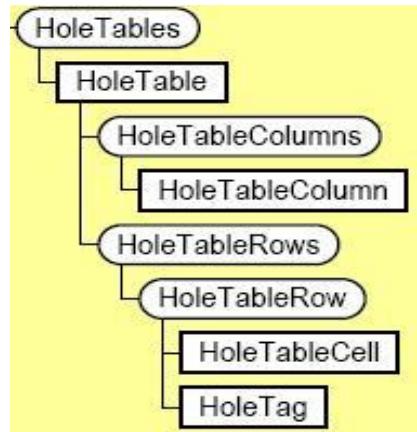
# Drawing ドキュメント

## 履歴テーブル／穴テーブル

変更履歴				
ZONE	REV	DESCRIPTION	DATE	APPROVED
1	1	<入力>	2006/07/20	Saito



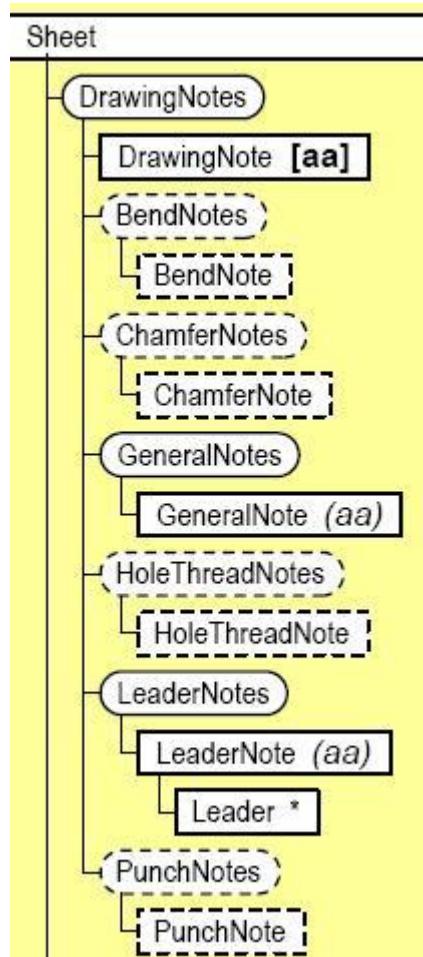
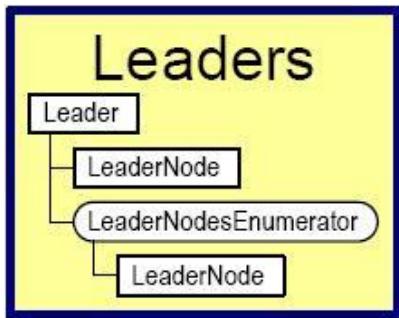
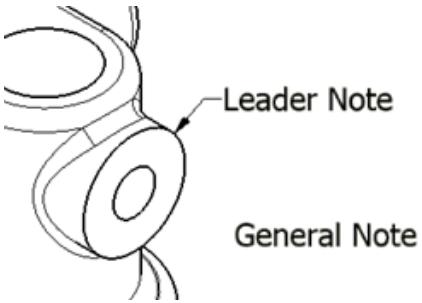
穴テーブル		
穴	X座標	Y座標
A1	13.22	10.2
B1	30.48	10.45
C1	31.74	24.31
D1	11.96	25.31



- シートに新規履歴テーブルの配置
- 既存履歴テーブルへの問い合わせ
- 既存列の編集と新規列の追加
- 関連した穴タグへのアクセス

# Drawing ドキュメント

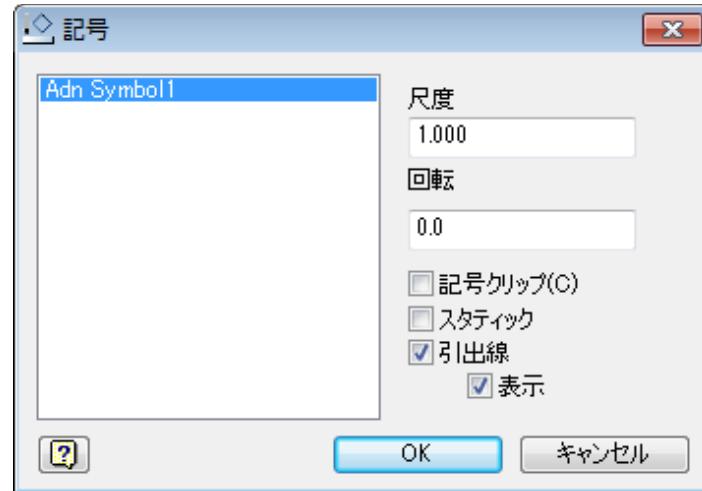
## 注記



- スケッチテキストと同じく、XMLによるフォーマットを使用
- 一般注記
  - ✓ シート上に配置されたテキスト。
- 引き出し注記
  - ✓ 引き出しを伴ったテキスト。
  - ✓ *GeometryIntent*を使用して引き出し点の位置を定義

# Drawing ドキュメント

## スケッチシンボル

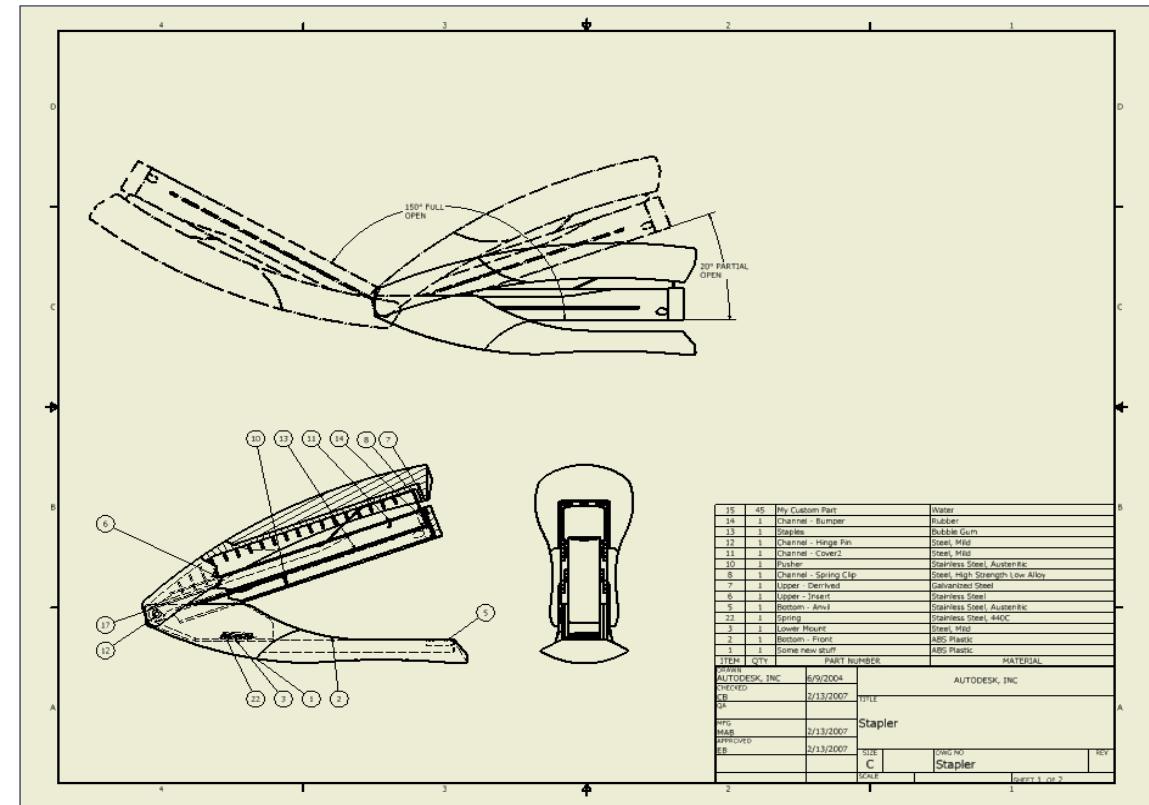
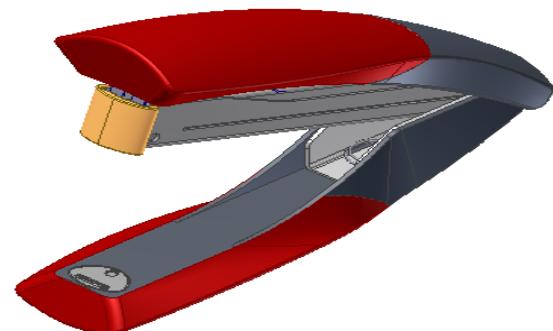
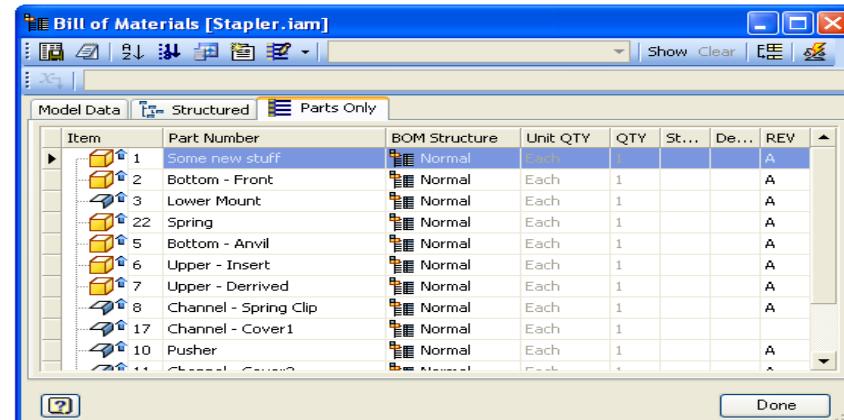


- スケッチシンボル定義
  - ✓ スケッチとして定義済み
  - ✓ プロンプトテキストのサポート
- スケッチシンボルのインスタンス
  - ✓ スケッチシンボル定義のインスタンス
  - ✓ 配置と編集時に、スケールと回転が可能
  - ✓ 引き出し線の作成が可能。GeometryIntentを使用して引き出し点の位置を定義

# Drawing ドキュメント

## 図面上のBOM

- 図面上のBOMへのアクセス
- バルーンまたはパーティリストから参照されているコンポーネントへのアクセス
- パーティリスト(エクスポート、回転、コピー等)

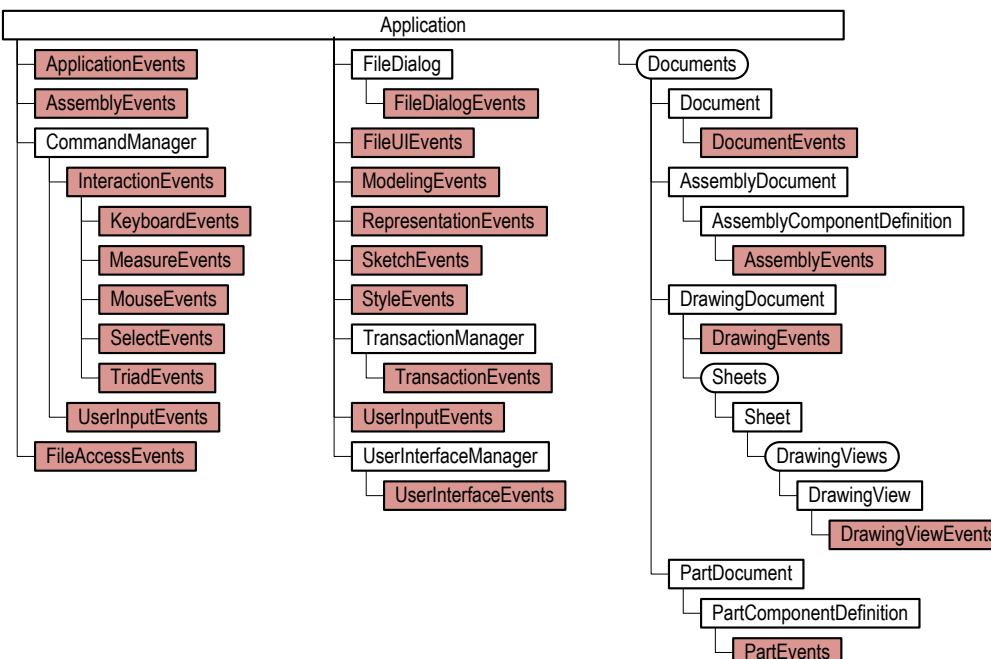




イベント

# イベント

## Inventor イベント



- イベントは、特定のアクションがInventor内で発生したときに、Inventor が送る通知
- イベントにより、Inventorで発生した事象に対応するプログラムが作成可能
- アクションに応答して情報を更新することで、独自の処理を作成が可能
- エンドユーザーとの高度なインタラクションのためのメカニズムを提供

# イベント

## イベントの基本ルール

```
Private Sub oAppEvents_OnSaveDocument(ByVal DocumentObject As Document, _
    ByVal BeforeOrAfter As EventTimingEnum, _
    ByVal Context As NameValueMap, _
    HandlingCode As HandlingCodeEnum)

End Sub
```

- 最初の引数は、そのイベントのコンテキストで有用な情報を提供
- *BeforeOrAfter* は、そのイベントがアクションの前／後のどちらで発生しているかを表す（いくつかのイベントは、前のみや後のみに発生し、ほとんどの場合は前後の両方で発生）
- *Context* は、そのイベントと関連付けられた、その他の情報を提供。それぞれのイベントに特有で、ほとんどのイベントで使用されない
- *HandlingCode* は、Inventorにどのようにイベントが取り扱われるかを指定する。Inventorの代わりにイベントを実行する/イベントをキャンセルする等を指定（ほとんどのイベントで未サポート）

# イベント

## イベントの使用(VBA)

```
Private WithEvents oAppEvents As ApplicationEvents

Private Sub UserForm_Initialize()
    Set oAppEvents = ThisApplication.ApplicationEvents
End Sub

Private Sub oAppEvents_OnSaveDocument(ByVal DocumentObject As Document, _
    ByVal FullDocumentName As String, _
    ByVal BeforeOrAfter As Inventor.EventTimingEnum, _
    ByVal Context As Inventor.NameValueMap, _
    ByRef HandlingCode As Inventor.HandlingCodeEnum)
    If BeforeOrAfter = kBefore Then
        MsgBox "Document is being saved."
    End If
End Sub
```

# イベント

## イベントの使用(VB.net)

```
Private m_ApplicationEvents As Inventor.ApplicationEvents

Public Sub Activate(ByVal addInSiteObject As Inventor.ApplicationAddInSite, _
                     ByVal firstTime As Boolean) _
    Implements Inventor.ApplicationAddInServer.Activate

    m_inventorApplication = addInSiteObject.Application
    m_ApplicationEvents = m_inventorApplication.ApplicationEvents

    AddHandler m_ApplicationEvents.OnOpenDocument, _
               AddressOf Me.m_ApplicationEvents_OnOpenDocument

End Sub

Private Sub m_ApplicationEvents_OnOpenDocument(ByVal DocumentObject As Inventor._Document, _
                                               ByVal FullDocumentName As String, _
                                               ByVal BeforeOrAfter As Inventor.EventTimingEnum, _
                                               ByVal Context As Inventor.NameValueMap, _
                                               ByRef HandlingCode As Inventor.HandlingCodeEnum)

    System.Windows.Forms.MessageBox.Show("OnOpenDocument: " + DocumentObject.DisplayName)

End Sub
```

# イベント

## イベントの使用(C#)

```
private Inventor.ApplicationEvents m_appEvents;

public void Activate(Inventor.ApplicationAddInSite addInSiteObject, bool firstTime)
{
    m_inventorApplication = addInSiteObject.Application;

    // ApplicationEventオブジェクトの取得
    m_appEvents = m_inventorApplication.ApplicationEvents;

    // OnSaveDocumentイベントハンドラの追加
    m_appEvents.OnSaveDocument += 
        new ApplicationEventsSink_OnSaveDocumentEventHandler(m_appEvents_OnSaveDocument);
}

// OnSaveDocumentイベントハンドラ
void m_appEvents_OnSaveDocument(_Document DocumentObject,
                                 EventTimingEnum BeforeOrAfter,
                                 NameValueMap Context,
                                 out HandlingCodeEnum HandlingCode)
{
    // C#では、output パラメータを指定する必要あり
    HandlingCode = HandlingCodeEnum.kEventNotHandled;
}
```

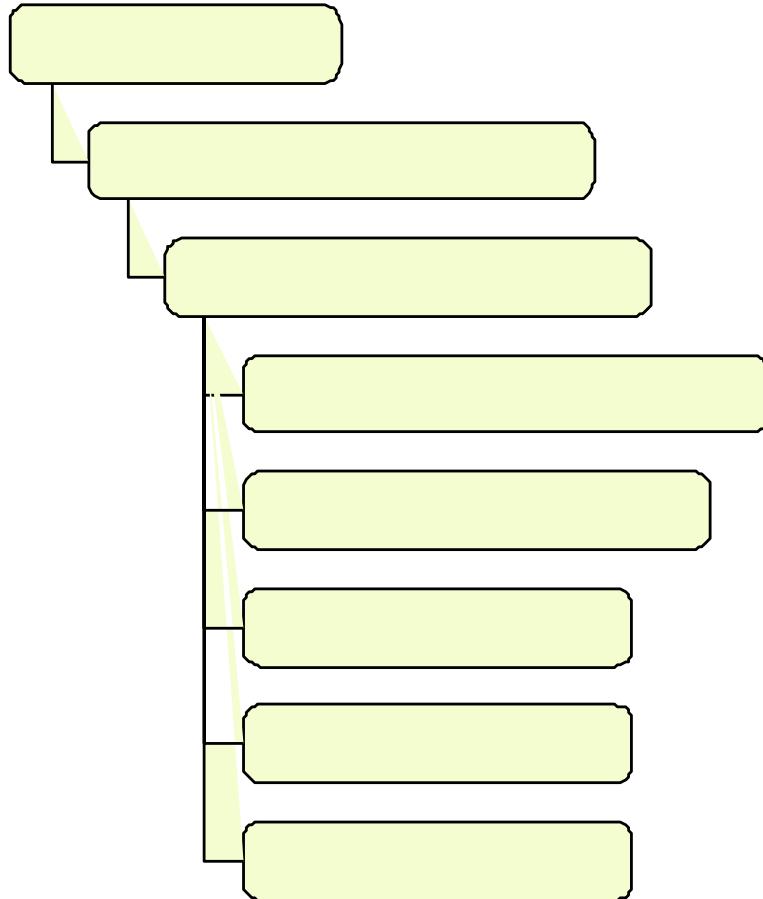
# イベント

## Interaction Events

- 標準Inventorコマンドの挙動を提供
  - ✓ *InteractionEvents*の開始により、実行中のコマンドのキャンセル
  - ✓ エンドユーザインタラクションにフォーカス
  - ✓ エンドユーザが別のコマンドを開始した場合実行中の*InteractionEvents*オブジェクトをキャンセル
- 選択ロジックのサポート
- インタラクティブなプレビューグラフィクスのサポート

# イベント

## Interaction Events



- *InteractionEvents* オブジェクトは、*CommandManager*の *CreateInteractionEvents* メソッドにより作成
- 作成された *InteractionEvents* オブジェクトは、現在アクティブなドキュメントと関連付けられる
- *InteractionEvents* オブジェクトは *Start* メソッドが呼ばれない限り Inventor の挙動への影響は発生しない

# イベント

## Interaction Events ワークフロー

1. *InteractionEvents*オブジェクトを作成
2. 必要なイベントに接続
3. 必要な挙動を実現するために、イベントセットにプロパティを設定
4. *InteractionEvents*オブジェクトを開始
5. イベントの発生を待機し、イベントに応答
6. *OnTerminate*イベントを監視しつつ、コマンド完了時に*InteractionEvents*を停止

# イベント

## モードレスダイアログ サンプルコード(VB.net)

```
Public Class WindowWrapper
    Implements System.Windows.Forms.IWin32Window

    Private _hwnd As IntPtr

    Public Sub New(ByVal handle As IntPtr)
        _hwnd = handle
    End Sub

    Public ReadOnly Property Handle() As IntPtr
        Implements System.Windows.Forms.IWin32Window.Handle
        Get
            Return _hwnd
        End Get
    End Property
End Class

Private Sub DisplayAsChildDlg()
    Dim myForm As New CustomForm
    myForm.Show(New WindowWrapper(m_inventorApplication.MainFrameHWND))
End Sub
```

- ダイアログを表示する際、デフォルトでは Inventorのメインウィンドウと独立して表示されてしまうため、以下のような問題が発生
  - ✓ InventorのWindowが前面に表示されてダイアログ隠されてしまう。
  - ✓ Inventorを最小化した際に、ダイアログが表示され続けてしまい、キーボードショートカットのキー入力がInventorに横取りされてしまう。
- ダイアログをInventor子Windowとして作成することで、この問題を回避することができる。サンプルコードは WindowWrapperユーティリティクラスによる実装

# イベント

## Pickメソッド

```
Public Sub GetSingleSelection()  
  
    'ユーザによるフィーチャー選択  
    Dim oObject As Object  
    oObject = m_inventorApplication.CommandManager.Pick( _  
        SelectionFilterEnum.kPartFeatureFilter  
        , "Pick a feature")  
  
    MsgBox( "Picked: " & oObject.Name)  
  
End Sub
```

- *CommandManager.Pick*
- 簡易化されたエンティティの選択
- *InteractionEvents*オブジェクトの外でのPre-selectおよびselectイベント
- エンティティの選択はSelectSetまたはInteractionEventsにより行われる。SelectSetは選択対象のコントロールがなく、object-actionワークフローが必要となる。一方で、InteractionEventsは完全な制御と柔軟性を提供するが実装難易度が高い。新しいPickメソッドは、InteractionEventsと同じワークフローを提供し、かつ実装が容易。InteractionEventsと比較すると制御と柔軟性は若干劣るもののはどんどの場合十分な機能を持つ。イベントの代わりに、関数呼び出しで、指定されたタイプの選択されたエンティティを戻す

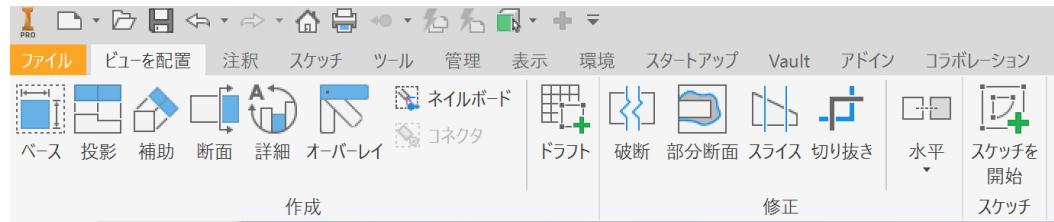


ユーザインターフェース

# ユーザインターフェース

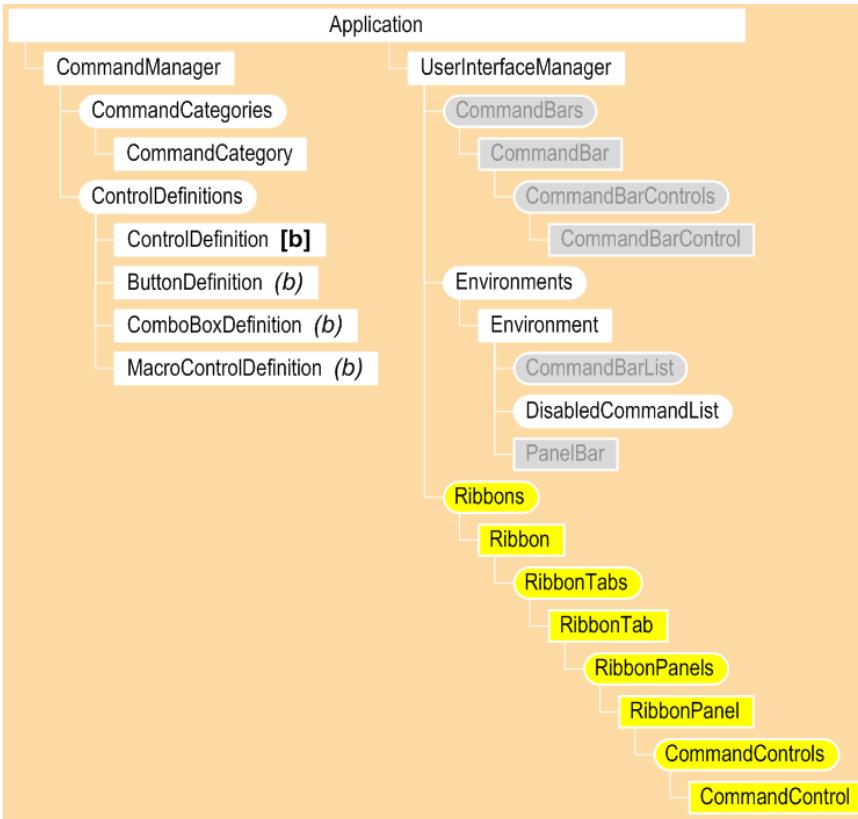
## リボンインターフェース

- リボンはInventorのデフォルトのユーザインターフェイス
- Microsoft 'WPF'テクノロジーに基づいた新しいインターフェース
- 既存のメニューとパネルバーを置き換える
- 頻繁に使われるコマンドを格納する新しいクイックアクセスツールバー



# ユーザインターフェース

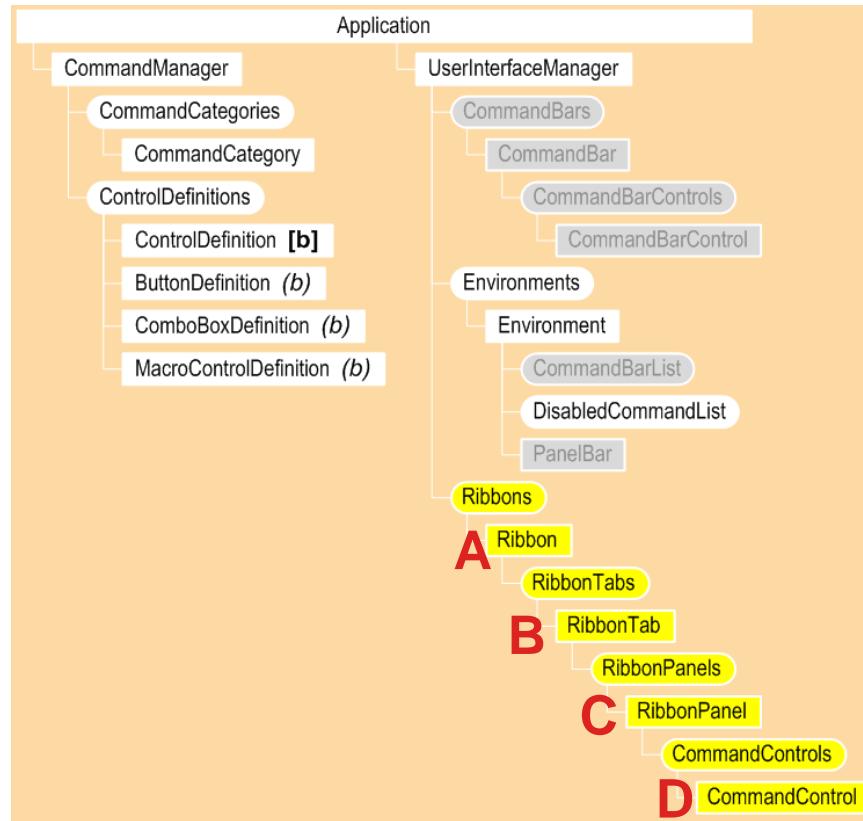
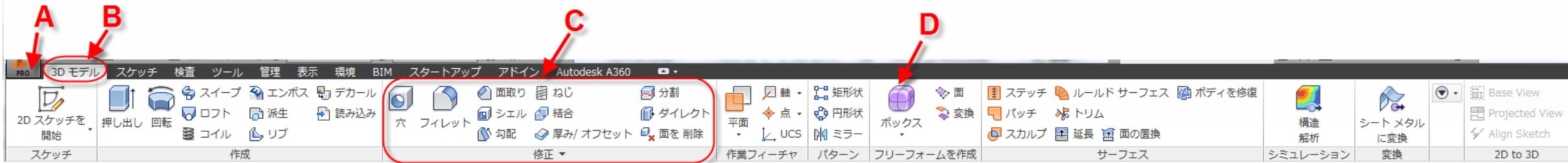
## リボンAPI オブジェクトモデル



- Inventor 2010よりAPIによるリボンをサポート
- *ButtonDefinition*オブジェクトにより、ボタンの見た目と挙動を定義
- リボンに関連するオブジェクトは、*UserInterfaceManager*オブジェクトから取得
- リボンオブジェクトの作成は、アドインの*Activate*メソッドで実施

# ユーザインターフェース

## リボンGUIとオブジェクト



- A…リボン
- B…リボンタブ
- C…リボンパネル
- D…コマンドコントロール(アイテム)

# ユーザインターフェース

## リボンエレメント - リボン

- 全てのリボンフィーチャーのアクセスを提供するトップレベルのオブジェクト
- 7つの利用可能なリボン
  - ✓ ZeroDoc(ドキュメントが開かれていない場合のリボン)
  - ✓ パーツ
  - ✓ アセンブリ
  - ✓ 図面
  - ✓ プrezentation
  - ✓ iFeature
  - ✓ UnknownDocument(Notebook編集中または図面Viewの方向編集時)
- 新規リボンは作成不可、既存リボンは修正可能。

# ユーザインターフェース

## リボンエレメント - 7つのリボン名を出力 (VB.Net)

```
Sub PrintRibbonNames()

    Dim oUIMng As UserInterfaceManager
    oUIMng = _InvApplication.UserInterfaceManager

    Dim oRibbon As Ribbon
    For Each oRibbon In oUIMng.Ribbons
        Debug.Print(oRibbon.InternalName)
    Next

End Sub
```

# ユーザインターフェース

## リボンエレメント - タブ

- *RibbonTab*オブジェクトはコマンド構成のワークフローを提供
- タブの新規作成および既存のタブの編集が可能
- タブを非表示にする事が可能
- ユーザーは、タブの可視性を設定する事が可能
- コンテキストタブを作成が可能

# ユーザインターフェース

## リボンエレメント - タブ

- *RibbonTabs.Add ( DisplayName , InternalName, ClientId, [ TargetTabInternalName ],[ InsertBeforeTargetTab ],[Contextual] )*

- ✓ *DisplayName*…リボンタブの表示名
- ✓ *InternalName*…リボンタブを一意に識別する内部名。タブを取得する際に使用
- ✓ *ClientId*…クライアントを一意に識別する文字列。AddinのCLSIDの文字列表現を指定。例) “{C9A6C580-3817-11D0-BE4E-080036E87B02}”。ダミーの文字列やから文字列の指定は推奨されない
- ✓ *TargetTabInternalName*…オプション引数。指定した内部名を持つタブの次にタブを作成
- ✓ *InsertBeforeTargetTab*…オプション引数。新規タブを *TargetTabInternalName*で指定したタブの前に作成するかを指定。デフォルト値はFalse
- ✓ *Contextual*…オプション引数。コンテキストタブとするかを指定。デフォルト値はFalse。Falseが指定された場合タブは表示状態で追加され、Trueが指定された場合タブは非表示で追加される。また、コンテキストタブは(スケッチ環境のスケッチタブのように)現在の環境に基づいて表示され、永続的に表示されないことを表す色付きで表示される。コンテキストタブはEnvironment.AdditionalVisibleRibbonTabsに追加する必要することで適切なInventor環境で表示される

# ユーザインターフェース

## リボンエレメント – ZeroDocに新規タブを作成(VB.Net)

```
Sub AddNewRibbonTab()

    'ZeroDocを取得
    Dim ZeroDocRibbon As Ribbon
    ZeroDocRibbon = _InvApplication.UserInterfaceManager.Ribbons.Item("ZeroDoc")

    Dim MyZeroDocTab As RibbonTab

    Try
        MyZeroDocTab = ZeroDocRibbon.RibbonTabs.Item("id_ZDTab")
    Catch

        'Toolsタブの後ろにタブを追加
        MyZeroDocTab = ZeroDocRibbon.RibbonTabs.Add("My_ZeroDocTab", _
            "id_ZDTab", _
            "{5B9AA8AD-1D99-4957-8CB4-9870F23CBBFB}", _
            "id_TabTools", _
            False)
    End Try

    MyZeroDocTab.Visible = True

End Sub
```

# ユーザインターフェース

## リボンエレメント - パネル

- リボンパネルはタブ内のコマンドを構造化可する
- 新規パネルの作成および既存パネルの編集が可能
- パネルを非表示にする事が可能
- Slideoutコントロール(リボンボタンに▼を表示して、クリック時にコマンドを表示するコントロール)が可能
- ユーザーは再構成・コントロールの表示・フローディングパネルの作成が可能

# ユーザインターフェース

## リボンエレメント - パネル

- *RibbonPanels.Add ( DisplayName , InternalName, ClientId, [ TargetPanelInternalName ] , [ InsertBeforeTargetPanel ] )*
  - ✓ *DisplayName*…リボンパネルの表示名
  - ✓ *InternalName*…リボンパネルを一意に識別する内部名。パネル取得時に使用可能
  - ✓ *ClientId*…クライアントを一意に識別する文字列。AddinのCLSIDの文字列表現を指定。例) “{C9A6C580-3817-11D0-BE4E-080036E87B02}”。ダミーの文字列やから文字列の指定は推奨されない
  - ✓ *TargetPanelInternalName*…オプション引数。指定した内部名を持つパネルの後にパネルを作成。
  - ✓ *InsertBeforeTargetPanel*…オプション引数。新規パネルを *TargetPanelInternalName*で指定したパネルの前に作成するかを指定。デフォルト値はFalse

# ユーザインターフェース

リボンエレメント - アセンブリドキュメントのモデルタブの作業フィーチャー パネルの可視性をトグルで制御 (VB.Net)

```
Sub TogglePanel()

    Dim oAssemblyRibbon As Ribbon
    oAssemblyRibbon = _InvApplication.UserInterfaceManager.Ribbons.Item("Assembly")

    Dim oAssembleModelTab As RibbonTab
    oAssembleModelTab = oAssemblyRibbon.RibbonTabs.Item("id_TabModel")

    Dim oAssemblyPanels As RibbonPanels
    oAssemblyPanels = oAssembleModelTab.RibbonPanels

    '内部名でWorkFeaturesのRibbon panelを取得
    Dim oWorkFeatureTabPanel As RibbonPanel
    oWorkFeatureTabPanel = oAssemblyPanels.Item("id_PanelAssm_ModelWorkFeatures")

    If oWorkFeatureTabPanel.Visible Then
        oWorkFeatureTabPanel.Visible = False
    Else
        oWorkFeatureTabPanel.Visible = True
    End If

End Sub
```

# ユーザインターフェース

## リボンエレメント - コントロール

- CommandControlオブジェクトはパネル内のすべてのメニューを表す
- 内容はControlDefinition オブジェクトで定義
- “クラシック”の CommandBarControl オブジェクトのような振る舞いCan be invisible.
- 大小のアイコンの使用が可能 (16x16 および 32x32)
- 様々なタイプのボタンやコンボボックスが可能
- セパレータオブジェクトを追加することで、メニュー内にセパレータを定義可能

# ユーザインターフェース

## リボンエレメント - リボン名の抽出 (VB.Net)

```
'dump Ribbon info
Public Sub PrintRibbon()

    Dim oLogFile As String = "c:\temp\ribbon.txt"
    Dim objWriter As New System.IO.StreamWriter(oLogFile, True)

    Dim oControl As CommandControl
    Dim oRibbon As Ribbon
    For Each oRibbon In mApp.UserInterfaceManager.Ribbons
        objWriter.WriteLine("Ribbon: " & oRibbon.InternalName)
        Dim oTab As RibbonTab
        For Each oTab In oRibbon.RibbonTabs
            objWriter.WriteLine(" Tab: " & oTab.DisplayName &, " &
                oTab.InternalName &, Visible: " & oTab.Visible)
            Dim oPanel As RibbonPanel
            For Each oPanel In oTab.RibbonPanels
                objWriter.WriteLine(" Panel: " & oPanel.DisplayName &, " &
                    oPanel.InternalName &, Visible: " & oPanel.Visible)
                For Each oControl In oPanel.CommandControls
                    objWriter.WriteLine("Control: " & oControl.DisplayName &, " &
                        oControl.InternalName &, Visible: " & oControl.Visible)
                Next
            Next
        Next
    Next

    objWriter.Close()
End Sub
```

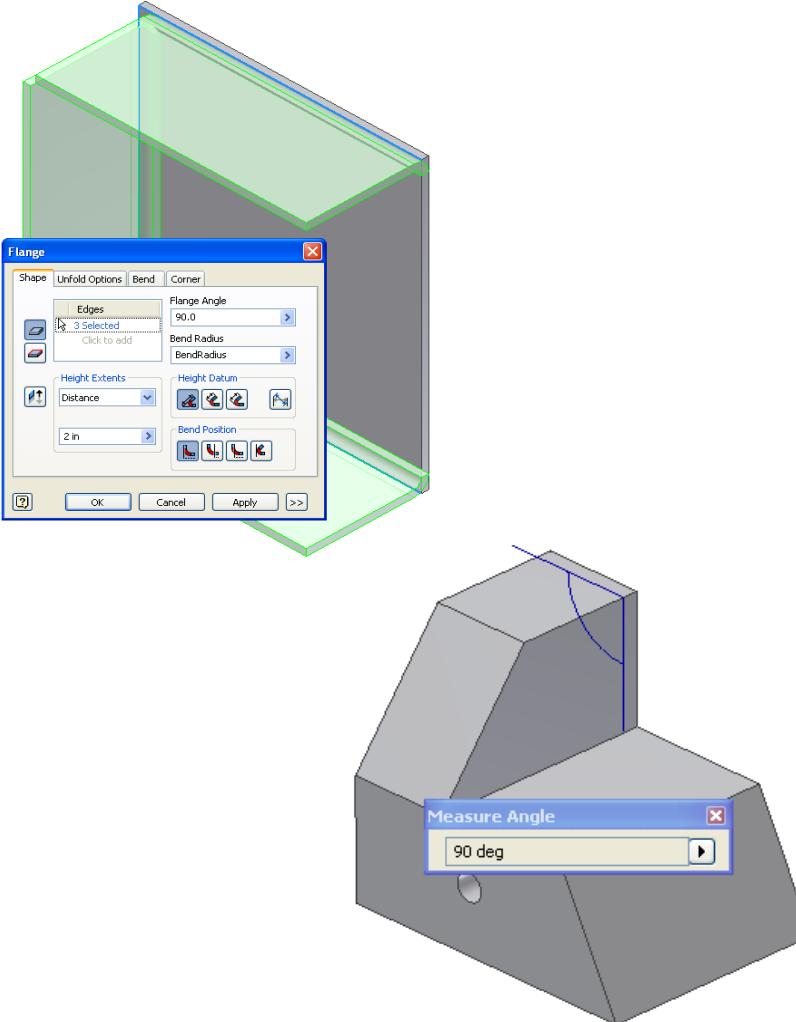
# ユーザインターフェース

## リボンでの考慮事項

- Add-InのActivateメソッドはリボンおよびクラシックUIをサポートする必要がある
  - *ControlDefinition* の作成は双方同じとなる
  - Activateメソッドで、FirstTime引数の値がTrueの場合、*UserInterfaceManager.InterfaceStyle* プロパティを確認してリボンまたはクラシックコントロールのどちらを作成するかを決定
- コマンドをGUIのどこに配置するかの考慮
  - ✓ 作成するコマンドが既存のコマンドの拡張機能と考えられる場合は、既存のパネルに追加
  - ✓ 作成するコマンドが特別なコマンドの場合は新しいパネルを作成
- 機能が利用可能な場合にタブ、パネル、コントロール表示されるような制御が利用可能
- Inventorの環境の考慮

# ユーザインターフェース

## エンドユーザから見たコマンド



- 複雑な機能を提供するコマンドではダイアログを用いてコマンドに必要となる入力を集める
- フィーチャなどを選択する挙動はコマンドに特異
- コマンドの実行結果をプレビュー
- 別のコマンド実行時に、実行中のコマンドを終了
- 例外的にViewコマンドは実行中のコマンドの実行を一時的に停止

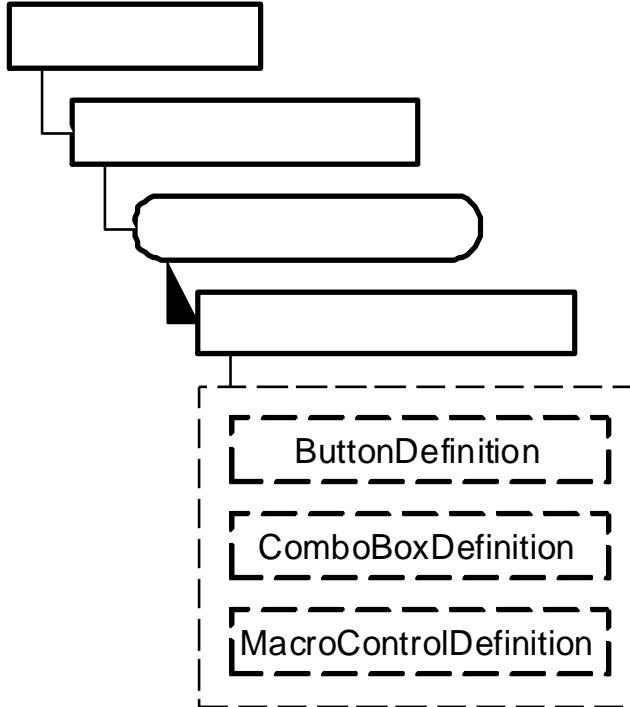
# ユーザインターフェース

## APIから見たコマンド

- コマンドは複数の要素により構成されている
- コマンドはAPI機能の組み合わせにより作成されている
  - GUI要素(ボタン、コンボボックス、その他)はエンドユーザに機能を実行する方法を提供
  - 実行中のコマンドの終了、ユーザの入力、動的なユーザへのビジュアルによるフィードバックなどのコマンドの挙動は、*InteractionEvents*オブジェクト、ダイアログおよびグラフィックスを用いて提供される。

# ユーザインターフェース

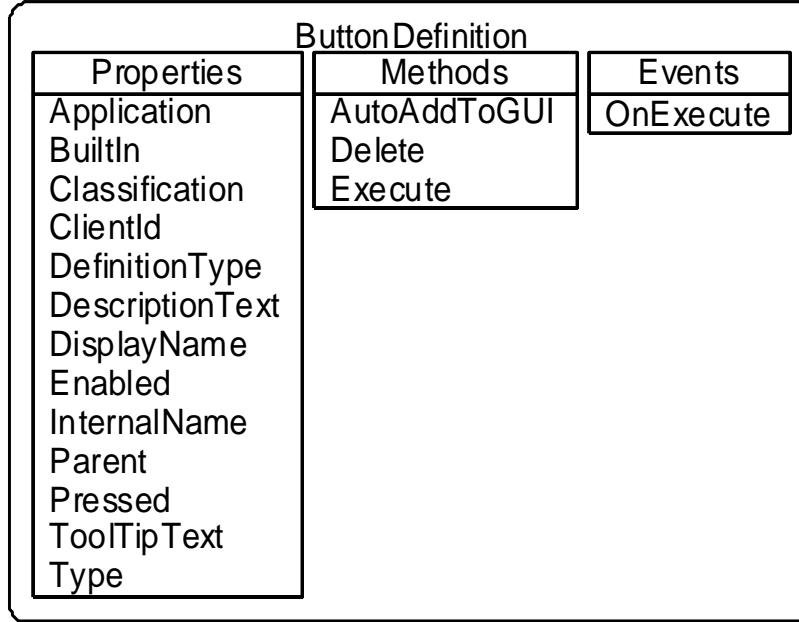
## ボタン及びその他のコントロール



- Inventorに表示されているボタンは二つのオブジェクトを作成することで定義される。
  - ControlDefinition* オブジェクト
  - CommandControl* オブジェクト
- ControlDefinition* オブジェクトは以下のコントロールのベースクラス
  - ✓ *ButtonDefinition* … 基本的なボタン
  - ✓ *ComboBoxDefinition* … コンボボックス
  - ✓ *MacroControlDefinition* … VBAのマクロを実行するボタン
  - ✓ その他 (ヘルプを参照)
- ControlDefinition* オブジェクトはすべてのInventorのビルトインコマンドにも存在している

# ユーザインターフェース

## *ControlDefinitions*と*Controls*



- *ControlDefinition*オブジェクトはコントロールがどのような外見なのか、またその挙動を定義。一方でUI上の何処にコントロールが存在するのかは定義しない
- *CommandControl* オブジェクトは、物理的なボタンまたはその他のコントロールを表す
- *CommandControl*オブジェクトは*ControlDefinition*オブジェクトを参照。*ControlDefinition*の変更は、関連するすべてのコントロールの更新を引き起こす

# ユーザインターフェース

## *ControlDefinition*の作成

- *ControlDefinitions.AddButtonDefinition* メソッド

✓ Sub AddButtonDefinition(

    DisplayName As String,

    InternalName As String,

    Classification As CommandTypeEnum,

    ByRef ClientId As [optional] VARIANT,

    ByRef DescriptionText As [defaultvalue("")] BSTR,

    ByRef ToolTipText As [defaultvalue("")] BSTR,

    ByRef StandardIcon As [optional] VARIANT,

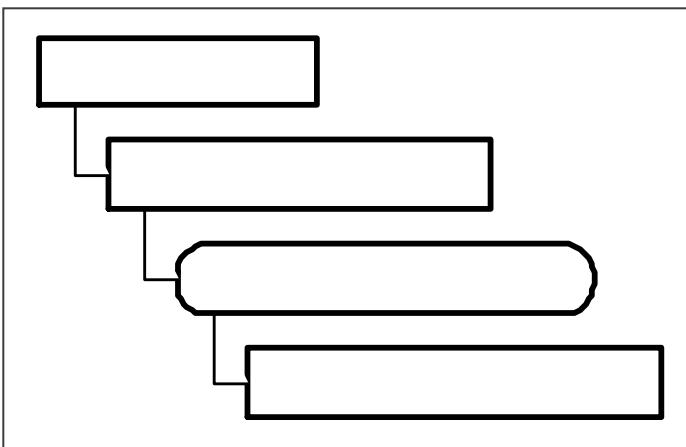
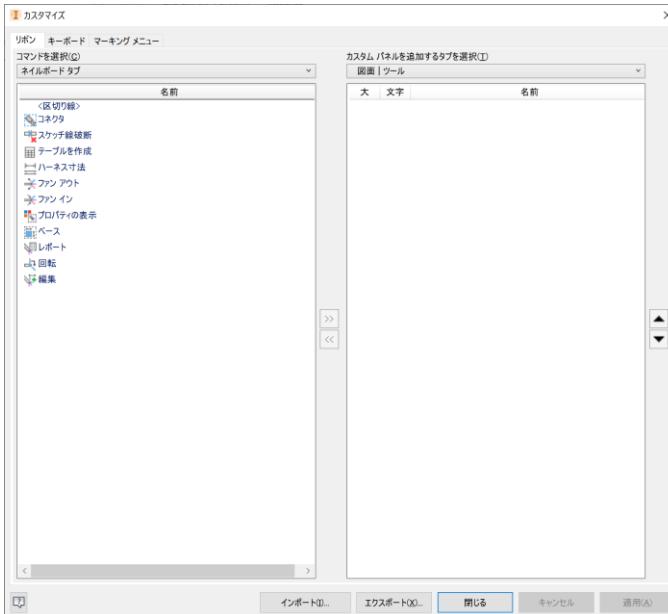
    ByRef LargeIcon As [optional] VARIANT,

    ByRef ButtonDisplay As ButtonDisplayEnum,

    Result As [out, retval] ButtonDefinition\*)

# ユーザインターフェース

## コマンドカテゴリー



- コマンドカテゴリを作成することで、コマンド (Control Definitions) をカスタマイズダイアログでグループ化することが可能

# ユーザインターフェース

## UI要素を作成するタイミング

- Add-InのActivateメソッドの*FirstTime*引数に注目
- *ApplicationAddInServer\_Activate( ByVal AddInSiteObject As ApplicationAddInSite, ByVal FirstTime As Boolean)*
  - ✓ *FirstTime*引数はAdd-Inが最初にアクティベートされた際にTrueとなる
  - ✓ Add-Inがアクティベートされるまに、*ControlDefinition*オブジェクトを作成
  - ✓ *FirstTime*引数の値がTrueの場合にのみコマンドバー、コマンドバーコントロール及び 環境情報を設定
  - ✓ Add-inが初回に開始した後は、InventorはUIの変更を記憶しており再作成をする必要はない

# ユーザインターフェース

## ボタンおよびリボンの作成サンプル (VB.Net)

```
Public Sub Activate(. . .) Implements Inventor.ApplicationAddInServerActivate
    Dim oCtrlDefs As ControlDefinitions = m_inventorApplication.CommandManager.ControlDefinitions
    mButtonDef = oCtrlDefs.AddButtonDefinition("Custom Cmd", "Company:AddInName:Buton1", _
        CommandTypeEnum.kQueryOnlyCmdType, "AddInGuid", _
        "Description", "Tolltip text", Nothing, Nothing, _
        ButtonDisplayEnum.kDisplayTextInLearningMode)

    Dim oUIMng As UserInterfaceManager = InvApplication.UserInterfaceManager
    If oUIMng.InterfaceStyle = InterfaceStyleEnum.kRibbonInterface Then
        'Fileコントロールにコマンドを追加
        Dim fileControls As CommandControls = oUIMng.FileBrowserControls
        fileControls.AddButton(mButtonDef)

        'アセンブリリボンの取得
        Dim assemblyRibbon As Inventor.Ribbon = oUIMng.Ribbons.Item("Assembly")
        'アセンブリタブを取得
        Dim assemblyTab As Inventor.RibbonTab = assemblyRibbon.RibbonTabs.Item("id_TabAssemble")
        'アセンブリタブに新規パネルを作成
        Dim panel As Inventor.RibbonPanel = assemblyTab.RibbonPanels.Add("Custom commands", _
            "Company:AddInName:Panel1", "AddInGuid")
        panel.CommandControls.AddButton(mButtonDef, True)
    Else
        'Perform classic UI initialisation
    End If
    AddHandler mButtonDef.OnExecute, AddressOf Me.mButtonDef_OnExecute 'Add handler VB.Net style
End Sub
```

# ユーザインターフェース

## ボタンにアイコンを使用

- ビットマップおよびアイコンはリソースに保存が可能
  - ✓ 以下のサンプルの場合、MyIcon.icoはエンベデッドリソース

```
Dim myStream As System.IO.Stream =  
    Me.GetType().Assembly.GetManifestResourceStream("MyAddIn.MyIcon.ico")
```

- ボタン定義を作成する際に、入力として使用
- InventorはIPictureDispオブジェクトを入力とする
- .Netは、新しいビットマップに新しいイメージタイプを使用

```
Dim myIcon As Icon = New Icon(myStream)
```

# ユーザインターフェース

## ボタンにアイコンを使用

- イメージ(またはアイコン)をIPictureDispオブジェクトに変換(VB.net)

```
Friend Class AxHostConverter
    Inherits AxHost
    Private Sub New()
        MyBase.New("")
    End Sub
    Public Shared Function ImageToPictureDisp(image As Image) As stdole.IPictureDisp
        Return DirectCast(GetIPictureDispFromPicture(image), stdole.IPictureDisp)
    End Function
    Public Shared Function PictureDispToImage(pictureDisp As stdole.IPictureDisp) As Image
        Return GetPictureFromIPicture(pictureDisp)
    End Function
End Class
```

```
Dim oStream1 As System.IO.Stream =
Me.GetType().Assembly.GetManifestResourceStream("ZeroRibbon.Tools.ico")

Dim oIcon1 As Icon = New Icon(oStream1)
Dim oIPictureDisp1 As Object = AxHostConverter.ImageToPictureDisp(oIcon1.ToBitmap())
```

- VB6での古い方法(Obsolete)

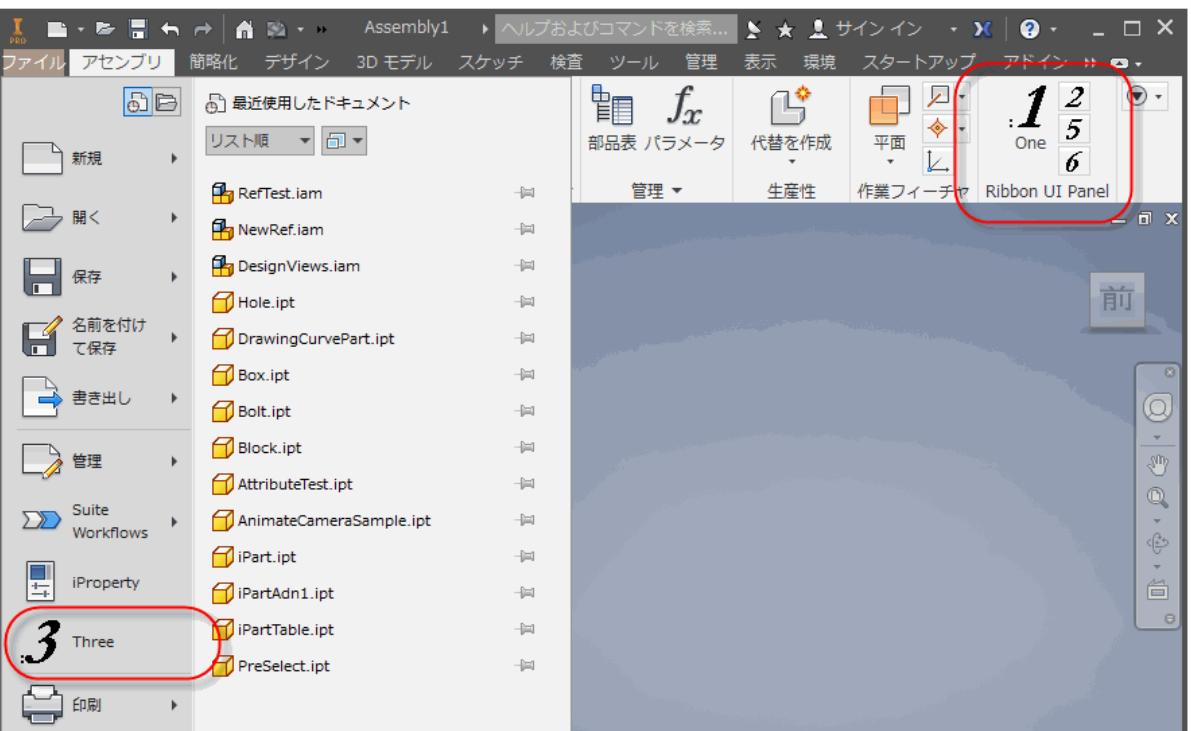
```
Dim myIPictureDisp As Object =
Microsoft.VisualBasic.Compatibility.VB6.Support.IconToIPicture(myIcon)
```

# 演習11

## 各種コントロール付 リボン UI の作成

- Inventorのアドインで、リボンインターフェースをサポートするUIを作成する。
- リボンUIの、「ファイル」コントロールに“ボタン3”を追加する。
- アセンブリリボンの”アセンブリ”タブに、”Ribbon UI Panel”パネルを追加し
  - ラージアイコン「ボタン1」
  - スモールのボタン「ボタン2、ボタン5、ボタン6」を追加

### ■ 演習のヒント





印刷とファイルトランスレーター

# 印刷とファイルトランスレーター

## 印刷

- *PrintManager* オブジェクトは様々なドキュメントからアクセスされる
- 図面とアペレンテスは追加の印刷機能を提供する特別な印刷マネージャーをサポート
- *PrintManager* オブジェクトはプリントコマンドダイアログで使用可能な設定と同じ設定を提供

PartDocument

PrintManager

AssemblyDocument

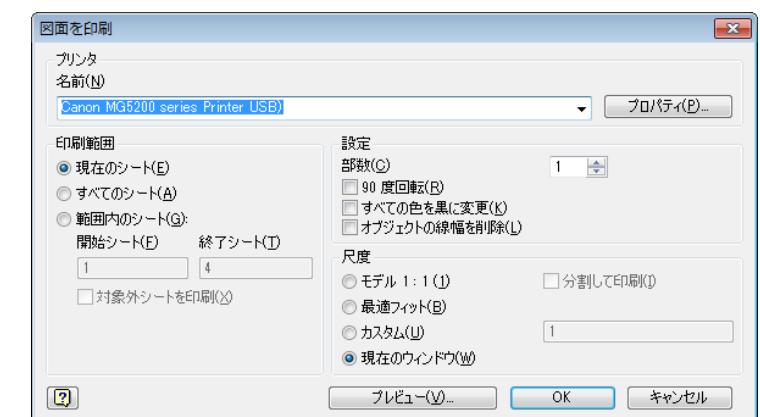
PrintManager

DrawingDocument

DrawingPrintManager

ApprenticeServerDocument

ApprenticeDrawing PrintManager



# 印刷とファイルトランスレーター

## *PrintManager*の主なプロパティ

- *NumberOfCopies*…デフォルト値1
- *PaperHeight / PaperWidth*…ペーパーの寸法設定。本プロパティは*PaperSize*プロパティにkPaperSizeCustomが設定された場合にのみ使用
- *Printer*…印刷デバイスの名前の取得/設定
- *PrintToFile(String FileName)*…指定されたファイルを現在の印刷設定で印刷
- *SubmitPrint*…現在のプロパティ設定を使用して印刷

# 印刷とファイルトランスレーター

## *DrawingPrintManager*

- *DrawingPrintManager*オブジェクトは *PrintManager* の派生クラスであり、*PrintManager*と同じプロパティ/メソッドと *DrawingPrintManager*で追加された固有のプロパティ/メソッドを持つ
  - ✓ *AllColorsAsBlack* … 図面のすべての色を印刷で黒にする必要があるかどうかを取得および設定
  - ✓ *PrintRange* … 印刷するシートに関する情報を取得および設定。 *kPrintCurrentSheet*、*kPrintAllSheets*、*kPrintSheetRange*のいずれかを設定
  - ✓ *ScaleMode* / *Scale* … 印刷の尺度を取得および設定
  - ✓ *RemoveLineWeights* … 図面の線の太さを印刷では削除するかを取得および設定
  - ✓ *GetSheetRange* / *SetSheetRange* … 印刷するシートの範囲を取得/設定するメソッド

# 印刷とファイルransレーター

## サンプル(VB.net)

- 最初にプリンターをセットしていることに注意。これにより、プリンターのデフォルト値の全てのオプションをリセットされる

```
Public Sub PrintSample()

    Dim oDrawDoc As DrawingDocument
    oDrawDoc = _InvApplication.ActiveDocument

    Dim oPrintMgr As DrawingPrintManager
    oPrintMgr = oDrawDoc.PrintManager

    With oPrintMgr
        .Printer = "Adobe PDF"
        .NumberOfCopies = 1
        .ScaleMode = PrintScaleModeEnum.kPrintBestFitScale
        .PaperSize = PaperSizeEnum.kPaperSizeA4
        .SubmitPrint()
    End With

End Sub
```

# 印刷とファイルトランスレーター

## ファイルトランスレーター

- シンプルなトランスレーション

- ✓ *Document.SaveAs*および*Documents.Open*メソッドを用いて、InventorのI/Oでドキュメントをトランスレート
- ✓ Inventorはファイル名の拡張子を用いて使用するトランスレータを決定
- ✓ デフォルト値の使用。デフォルトの設定は前回使用したインタラクティブなファイルトランスレーションの設定

*Documents.Open( "C:\Temp\Test.stp" )*

*Document.SaveAs( "C:\Temp\New.dwg", True )*

# 印刷とファイルトランスレーター

## ファイルトランスレーター

- ほとんどのInventor のトランスレーターは、トランスレーターアドインとして実装されている
- トランスレーターアドインのインターフェースを通して、直接トランスレーターにアクセスおよび実行が可能
- APIにより、トランスレーターの様々なオプションの制御が可能

# 印刷とファイルトランスレーター

## トランスレーターの聞くオプション サンプル(VB.net)

```
Public Sub GetTranslatorOpenOptions()

    Dim clsId As String
    clsId = "{90AF7F40-0C01-11D5-8E83-0010B541CD80}" 'STEP Translator

    Dim oTranslator As TranslatorAddIn
    oTranslator = _InvApplication.ApplicationAddIns.ItemById(clsId)

    Dim medium As DataMedium
    medium = _InvApplication.TransientObjects.CreateDataMedium
    medium.FileName = "C:\Temp\MyInventorDoc.xxx"
    medium.MediumType = MediumTypeEnum.kFileNameMedium

    Dim context As TranslationContext
    context = _InvApplication.TransientObjects.CreateTranslationContext

    Dim options As NameValueMap
    options = _InvApplication.TransientObjects.CreateNameValueMap

    Dim index As Integer
    If oTranslator.HasOpenOptions(medium, context, options) Then
        For index = 1 To options.Count
            Debug.Print(options.Name(index) & " = " & options.Value(options.Name(index)))
        Next
    End If

End Sub
```

# 印刷とファイルトランスレーター

## トランスレーターの保存オプション サンプル(VB.net)

```
Public Sub GetTranslatorSaveAsOptions()

    Dim clsId As String
    clsId = "{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}" 'DWF Translator

    Dim oTranslator As TranslatorAddIn
    oTranslator = _InvApplication.ApplicationAddIns.ItemById(clsId)

    Dim context As TranslationContext
    context = _InvApplication.TransientObjects.CreateTranslationContext
    context.Type = IOMechanismEnum.kUnspecifiedIOMechanism

    Dim options As NameValueMap
    options = _InvApplication.TransientObjects.CreateNameValueMap

    Dim SourceObject As Object
    SourceObject = _InvApplication.ActiveDocument

    Dim index As Integer
    If oTranslator.HasSaveCopyAsOptions(SourceObject, context, options) Then
        For index = 1 To options.Count
            Debug.Print(options.Name(index) & " = " & options.Value(options.Name(index)))
        Next
    End If

End Sub
```

# 印刷とファイルトランスレーター

## Inventor 2013の組み込みトランスレータアドイン

- DWF:{0AC6FD95-2F4D-42CE-8BE0-8AEA580399E4}
- PDF: {0AC6FD96-2F4D-42CE-8BE0-8AEA580399E4}
- DWFx:{0AC6FD97-2F4D-42CE-8BE0-8AEA580399E4}
- JT: {16625A0E-F58C-4488-A969-E7EC4F99CACD}
- i-drop Translator: {21DB88B0-BFBF-11D4-8DE6-0010B541CAA8}
- CATIA V5 Part Export:{2FEE4AE5-36D3-4392-89C7-58A9CD14D305}
- SolidWorks:{402BE503-725D-41CB-B746-D557AB83BAF1}
- Pro/ENGINEER: {46D96B7A-CF8A-49C9-8703-2F40CFBDF547}
- STL: {533E9A98-FC3B-11D4-8E7E-0010B541CD80}
- DWF Markup Manager:{55EB0FA-EF60-4028-A350-502CA148B499}
- Pro/ENGINEER Granite:{66CB2667-73AD-401C-A531-64EC701825A1}

# 印刷とファイルトランスレーター

## Inventor 2013の組み込みトランスレータアドイン

- IDF Translator: {6C5BBC04-5D6F-4353-94B1-060CD6554444}
- SAT: {89162634-02B6-11D5-8E80-0010B541CD80}
- CATIA V5 Product Export: {8A88FC01-0C32-4B3E-BE12-DDC8DF6FFF18}
- Pro/ENGINEER Neutral:{8CEC09E3-D638-4E8F-A6E1-0D1E1A5FC8E3}
- CATIA V5 Import:{8D1717FA-EB24-473C-8B0F-0F810C4FC5A8}
- Parasolid Text:{8F9D3571-3CB8-42F7-8AFF-2DB2779C8465}
- STEP: {90AF7F40-0C01-11D5-8E83-0010B541CD80}
- IGES: {90AF7F44-0C01-11D5-8E83-0010B541CD80}
- UGS NX: {93D506C4-8355-4E28-9C4E-C2B5F1EDC6AE}
- Content Center Item Translator: {A547F528-D239-475F-8FC6-8F97C4DB6746}
- Parasolid Binary:{A8F8F8E5-BBAB-4F74-8B1B-AC011251F8AC}

# 印刷とファイルトランスレーター

## Inventor 2013の組み込みトランスレータアドイン

- Drag & Drop Interoperability: {B95D705C-E915-4A5B-A498-E73AC98923A2}
- DWG: {C24E3AC2-122E-11D5-8E91-0010B541CD80}
- DXF: {C24E3AC4-122E-11D5-8E91-0010B541CD80}
- Alias:{DC5CD10A-F6D1-4CA3-A6E3-42A6D646B03E}

# 印刷とファイルトランスレーター

## DataIOオブジェクト

- フォーマット済みデータの入出力を行う汎用オブジェクト
- *GetInputFormats / GetOutputFormats* メソッドで、DataIOオブジェクトがサポートしているフォーマット入出力フォーマットを取得
  - ✓ スケッチ : DXF、DWGフォーマット
  - ✓ iProperties : FlatPattern、XML
  - ✓ worksurfaces : SAT

...

# 印刷とファイルトランスレーター

## DataIO サンプル(VB.net)

```
Public Sub ExportWorkSurface()

    Dim oWorkSurfaces As WorkSurfaces
    oWorkSurfaces = _InvApplication.ActiveDocument.ComponentDefinition.WorkSurfaces

    Dim oDataIO As DataIO
    oDataIO = oWorkSurfaces(1).SurfaceBodies(1).DataIO
    oDataIO.WriteDataToFile("ACIS SAT", "C:\Temp\result.sat")

End Sub

Public Sub ExportSketchDXF()

    Dim oSketch As PlanarSketch
    oSketch = _InvApplication.ActiveDocument.ComponentDefinition.Sketches(1)

    Dim oDataIO As DataIO
    oDataIO = oSketch.DataIO
    Call oDataIO.WriteDataToFile("DXF", "C:\Temp\dxfout.dxf")

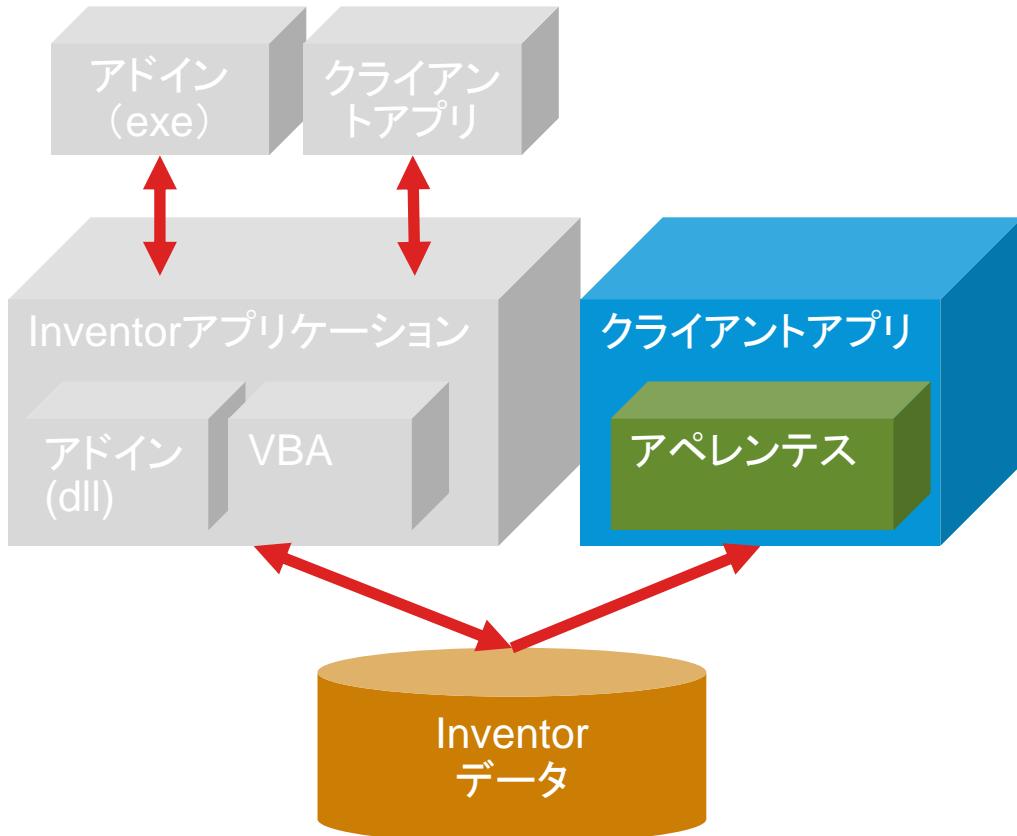
End Sub
```



アペレンテス

# アペレンテス

## アペレンテス



- アペレンテスはActiveXコンポーネント
- アペレンテスを使用するクライアントのインプロセスで動作
- アペレンテスはInventor APIのサブセットを提供
- スタンドアロンで動作 (Inventorのインストールは不要)
- 無償で利用可能 (Inventor Viewerの一部としてインストールされる)
- アペレンテスはプロセス外からInventorに接続する場合と比較して効率的に動作

# アペレンテス

## アペレンテスの機能

- アペレンテスにより以下に対して読み取りアクセスが可能
  - ✓ Assembly structure
  - ✓ B-Rep
  - ✓ Drawing sheets and views (limited access)
  - ✓ iParts
  - ✓ iAssemblies
  - ✓ BOM
- iProperties、アトリビュート、ファイルリファレンスへの読み取り及び書き込みアクセス

# アペレンテス

## 使用上の注意

- Inventorのプロセス内での利用は不可
  - ✓ Add-In内での利用は不可 (exe形式のアドインはOK)
  - ✓ InventorのVBAからの利用は不可
- Inventor 11以前では、ファイルリファレンスの編集にはアペレンテスが唯一の方法
- Inventor 2008以前では、アペレンテスの機能はアペレンテス自身のタイプライブラリで提供
- 現在では、Inventorのタイプライブラリにはアペレンテスのすべての機能が含まれており、アペレンテスプロジェクトでも、Inventor.Interopへの参照を設定する

# アペレンテス

## アペレンテスとInventorの違い

- “new ApprenticeServerComponent”でインスタンスを作成
- アペレンテスにはドキュメントコレクションは存在しない
- Documentオブジェクト
  - ✓ *ApprenticeServerDocument (\*.ipt & \*.iam)*
  - ✓ *ApprenticeServerDrawingDocument (\*.idw)*
- アペレンテスでは、以前のバージョンのInventorファイルを保存することは出来ない。アペレンテスで保存をする前に、Inventorで開いて、保存することでファイルをマイグレーションする必要がある
- Inventor 2022の新機能モデル状態へアクセスするAPIは、アペレンテスでは提供されず、アペレンテスのAPIでは、最後にInventorファイルが保存された際にアクティブなモデル状態へアクセスをする

# アペレンテス

## アペレンテスのサンプル (VB.net)

```
Private Sub ApprenticeSample()

    ' Apprenticeの作成
    Dim oApprentice As ApprenticeServerComponent
    oApprentice = New ApprenticeServerComponent

    ' ドキュメントを開く
    Dim oDoc As ApprenticeServerDocument
    oDoc = oApprentice.Open("C:\Temp\Part.ipt")

    MsgBox("Opened: " & oDoc.DisplayName)

End Sub
```

# アペレンテス

## アペレンテスによるファイルの保存 (VB.net)

- iPropertiesのみを編集した場合は*FlushToFile*メソッドが、ドキュメントを書き戻さないため効率的な方法

```
Private Sub SetProperty(ByVal author As String)

    Dim oApprenticeDoc As ApprenticeServerDocument
    oApprenticeDoc = mApprenticeServer.Open("c:\Temp\MyPart.ipt")

    'Get "Inventor Summary Information" PropertySet
    Dim oPropertySet As PropertySet
    oPropertySet = oApprenticeDoc.PropertySets("{F29F85E0-4FF9-1068-AB91-08002B27B3D9}")

    'Get Author property
    Dim oProperty As Inventor.Property = oPropertySet.Item("Author")
    oProperty.Value = author

    oApprenticeDoc.PropertySets.FlushToFile()
    oApprenticeDoc.Close()

End Sub
```

# アペレンテス

## アペレンテスによるファイルの保存 (VB.net)

- そのほかの変更には`FileSaveAs`オブジェクトを使用

```
Private Sub SaveFileSample()

    Dim oApprentice As New ApprenticeServerComponent

    ' Open a document.
    Dim oDoc As ApprenticeServerDocument
    oDoc = oApprentice.Open("C:\Temp\Assembly1.iam")

    ' Do some change: iProperties and attributes and/or file references.

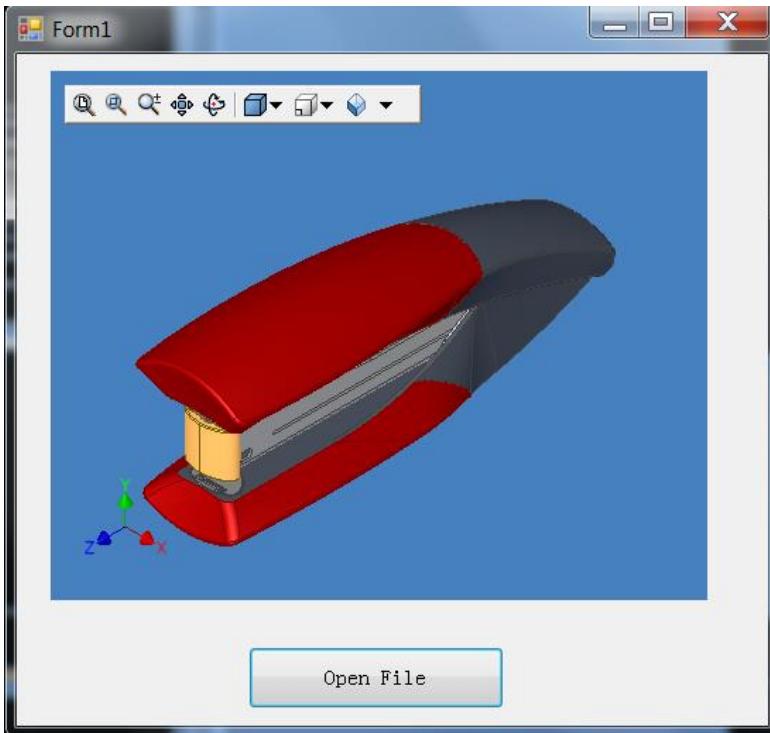
    ' Set a reference to the FileSaveAs object.
    Dim oFileSaveAs As FileSaveAs
    oFileSaveAs = oApprentice.FileSaveAs

    ' Save the file.
    Call oFileSaveAs.AddFileToSave(oDoc, oDoc.FullName)
    Call oFileSaveAs.ExecuteSave()

End Sub
```

# アペレンテス

## Inventor View コントロール



- Inventorファイルのビューウーコントロール
- インストールディレクトリ
  - ✓ <Inventor View Installation>\bin\InventorViewCtrl.ocx

※Inventor 2018以降32bit版のコントロールの提供を廃止し、64bit版のみを提供
- Visual Studioが32bitアプリケーションであるため、Visual Studioのフォーム編集画面等のGUIでコントロールを配置することが出来ないため、プログラムから動的にコントロールを追加する必要がある
- 無償 (Inventor Viewerと合わせて提供)
- 2012以降、レジストリフリーが導入されたためOCXを使用したい場合、  
プラグインの開発者はマニフェストファイルをリンクする必要がある。リンク方法は以下の2通り
  - ✓ 1. mt.exeを使用してマニフェストファイルを生成し、マニフェストファイルをアプリケーションにリンク
  - ✓ 2. 古いスタイル(レジストリ)を使用。コマンドプロンプトから管理者権限で“Regsvr32 InventorViewCtrl.ocx”を実行して、InventorViewCtrl.ocxを手動で登録



クライアントグラフィクス

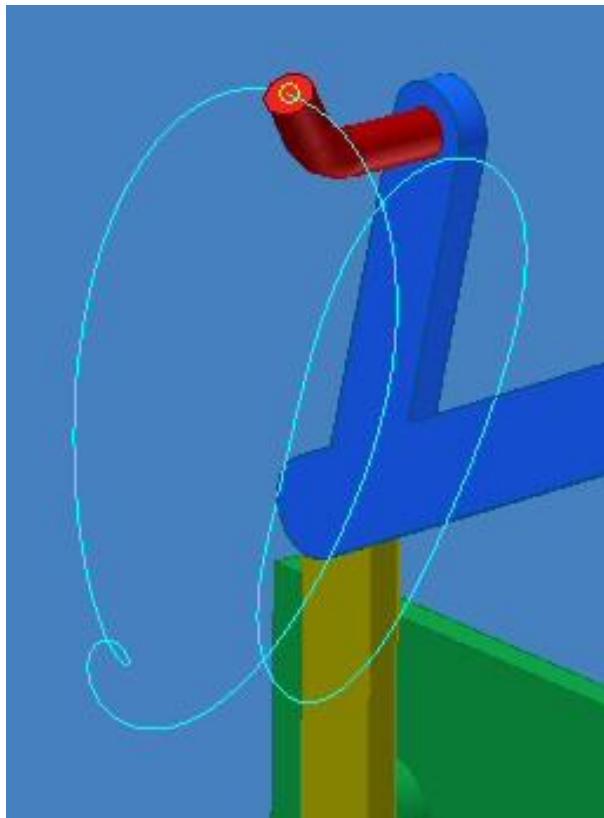
# クライアントグラフィクス

## 定義

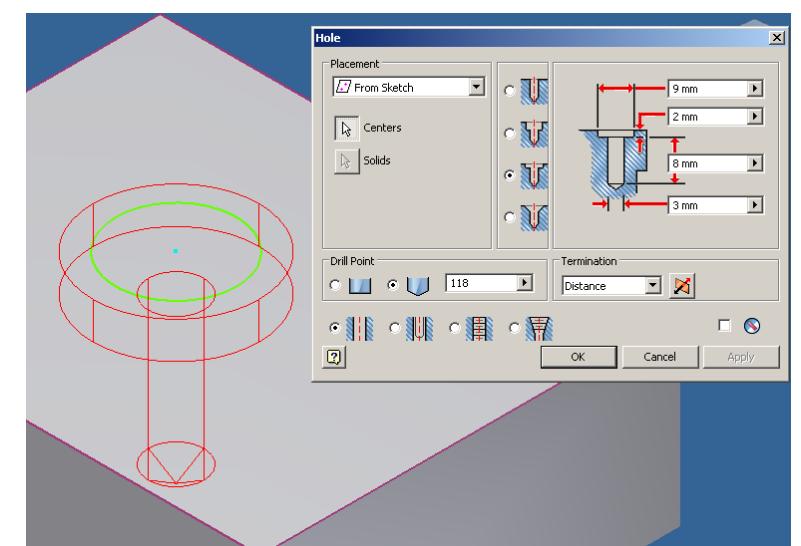
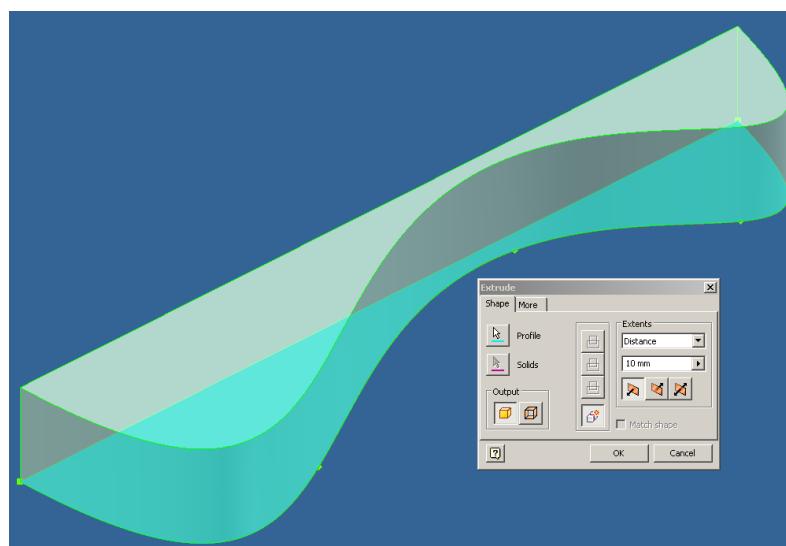
- Inventor APIはコンポーネント、図面ビュー上にグラフィクスを配置する手段を提供する。点、線、矩形、テキスト等のプリミティブをまとめて*ClientGraphics*として参照され、セッションの継続期間中Inventorにより管理・変形される
- その性質状、クライアントグラフィクスは、パーツやアセンブリと異なり一時的な物となる。しかしクライアントグラフィクスは画面のリフレッシュで破棄されない
- クライアントグラフィクスは、ズームやパンその他のビューの変更に応じて変形する

# クライアントグラフィクス

## 目的



- クライアントグラフィクスは開発者がエンドユーザにビジュアルな手がかりを提供することを可能にする
- Inventor自身多くのコマンドで行っている
  - 押し出し、穴の追加、スイープ等
- 開発者は、自身のアプリケーションで同様な手がかりを提供可能



# クライアントグラフィクス

なぜ必要なのか

- Inventorのネイティブグラフィクスに並べてカスタムグラフィクスを描画
- 特定のグラフィクスプラットフォームからの分離
- 視覚フィードバックとの統合によるプラグインコマンドの拡張
- OpenGLグラフィクスに基づくソリューションのマイグレーション

# クライアントグラフィクス

## クライアントグラフィクスの応用

- クライアントグラフィクスはInventorのNativeコマンドで広く使用されている
  - ✓ カスタムおよび一時的なオブジェクトの表示
  - ✓ コマンド中のインタラクティブなプレビューの作成
  - ✓ 様々な分析結果の表示
  - ✓ コマンド内でのカスタムマニピュレーターの作成
  - ✓ ユーザ体験の拡張

# クライアントグラフィクス

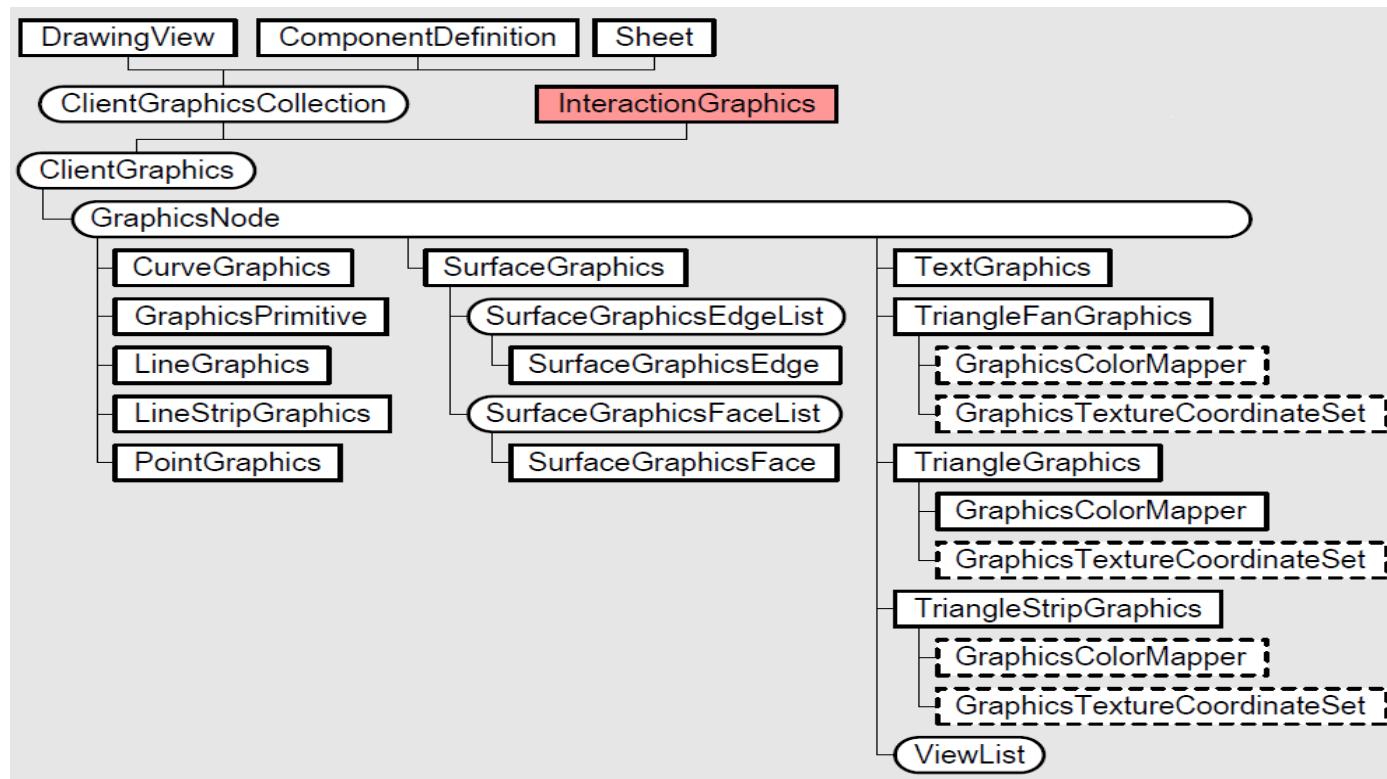
## オブジェクトモデル

- クライアントグラフィクスの作成は2つのステージのプロセスと考えら、それぞれに別のオブジェクトモデルが存在し、
  - ✓ 最初のステージはデータのセットアップ。最も重要なのは、クライアントグラフィクスが基準とする座標のリスト
  - ✓ 次のステージは、準備したデータを使用するグラフィクスピリミティブのセットアップとなる
- このデータのグラフィクスからの分離により、一つの一覧のデータが複数のプリミティブから参照されることを可能とする

# クライアントグラフィクス

## グラフィックオブジェクト

- *ClientGraphics*…グラフィクスグループのオーナー。表示・非表示 選択可否をコントロール
- *GraphicsNode*…グラフィクスピリミティブのオーナー。表示・非表示、選択可否、色及び位置を管理
- Graphic primitives…表示されるグラフィクス



# クライアントグラフィクス

## *GraphicsNode*の基本プロパティ

- *GraphicsNode*オブジェクトは内包するプリミティブグラフィクスのデフォルト動作をコントロールするプロパティを持つ
  - ✓ *Render Style*
  - ✓ *Visibility*  
*Visible* | *VisibleInViews* | *VisibleInActiveEditObject*
  - ✓ *Selectable*
  - ✓ *Transformation*

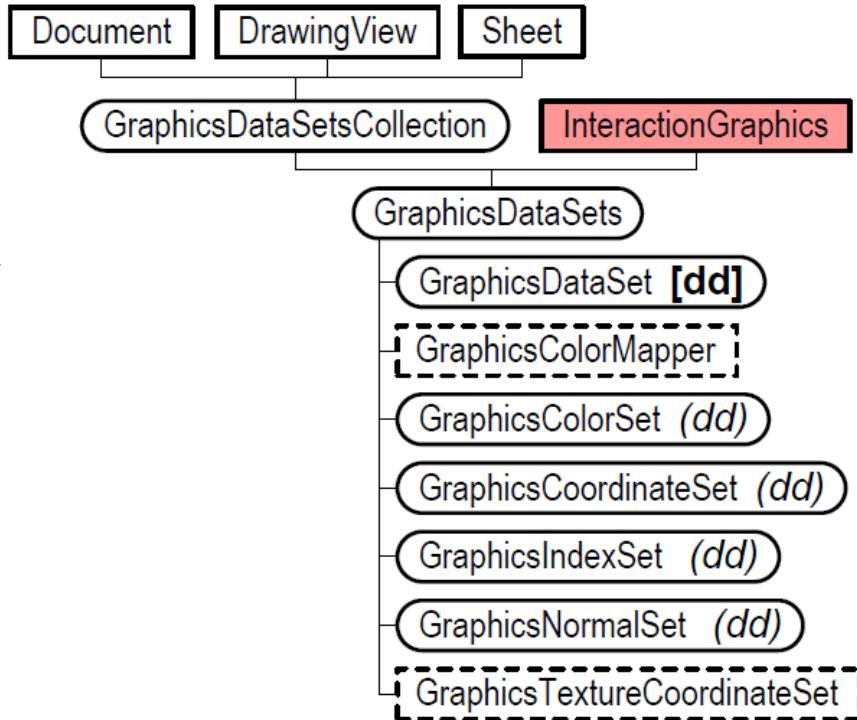
# クライアントグラフィクス

## *ClientGraphics*の種類

- 通常の*ClientGraphics*
  - ✓ ドキュメントに関連づけられる
  - ✓ *ClientFeature*により所有されるまでは一時的なオブジェクト
  - ✓ 異なるソースから取得可能
    - *PartComponentDefinition*
    - *AssemblyComponentDefinition*
    - *FlatPattern*
    - *DrawingView, Sheet*
    - *ClientFeatureDefinition*
- インタラクション*ClientGraphics*
  - ✓ *InteractionEvents* 実行中にのみ使用可能
  - ✓ コマンド実行中にプレビューまたはマニピュレーターを表示
  - ✓ *InteractionEvents* 終了時に自動的に削除される

# クライアントグラフィクス

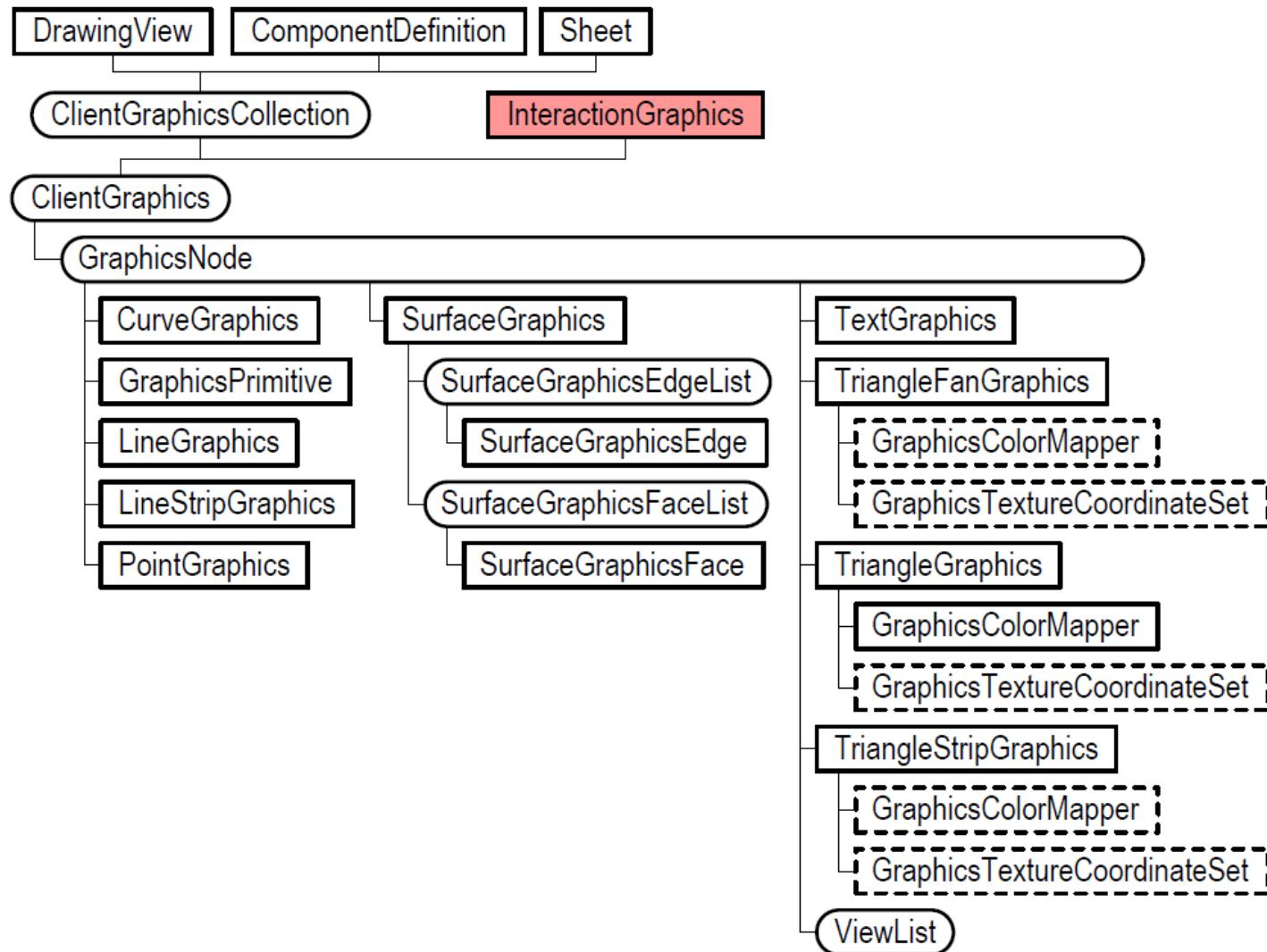
## *ClientGraphics*データのオブジェクトモデル



- データセットは、グラフィクスジオメトリー定義に使用される元データを保持する
  - ✓ *GraphicDataSets* … *GraphicsDataSet*オブジェクトのオーナーオブジェクト
  - ✓ *GraphicsColorSet* … 色セットを定義
  - ✓ *GraphicsCoordinateSet* … 座標点のセットを定義
  - ✓ *GraphicsIndexSet* … 他のデータセットへのインデックスを定義するLong値のセット
  - ✓ *GraphicsNormalSet* … 法線ベクトルとして使用される一連のベクトルのセットを定義

# クライアントグラフィクス

## *ClientGraphics*のオブジェクトモデル



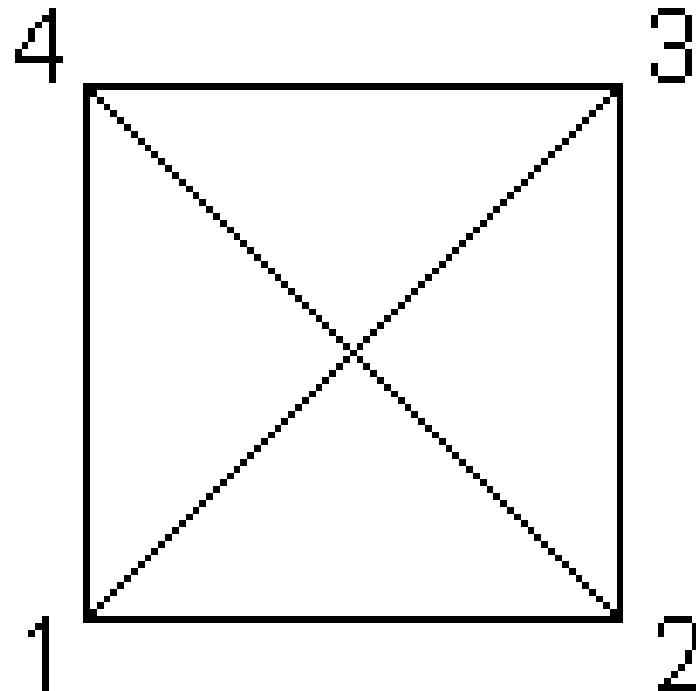
# クライアントグラフィクス

## *G*raphics*P*rimitivesの共通プロパティ

- *GP*rimitiveオブジェクトはInventorのGUI上に表示されるグラフィクスを表す
- 以下は、*GP*rimitiveオブジェクトで共通のプロパティとメソッド
  - ✓ BurnThrough
  - ✓ DepthPriority
  - ✓ SetViewSpaceAnchor
  - ✓ SetTransformBehavior

# クライアントグラフィクス

## LineGraphics



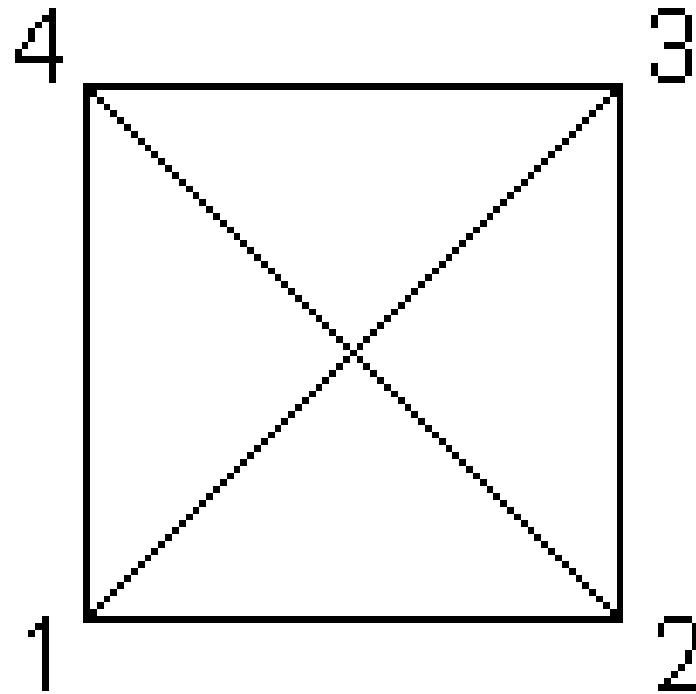
- *Public Function AddLineGraphics() As LineGraphics*
- 個々のラインセグメント

座標データ

$(0,0,0)-(1,0,0)$   
 $(1,0,0)-(1,1,0)$   
 $(1,1,0)-(0,1,0)$   
 $(0,1,0)-(0,0,0)$   
 $(0,0,0)-(1,1,0)$   
 $(1,0,0)-(0,1,0)$

# クライアントグラフィクス

## *LineStripGraphics*



- *Public Function AddLineStripGraphics() As LineStripGraphics*

- 接続された一連のライン

座標データ

(0,0,0),

(1,0,0),

(1,1,0),

(0,1,0),

(0,0,0),

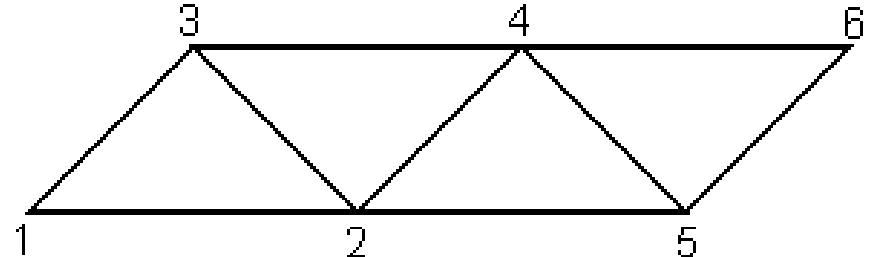
(1,1,0),

(1,0,0),

(0,1,0)

# クライアントグラフィクス

## *LineStripGraphics*



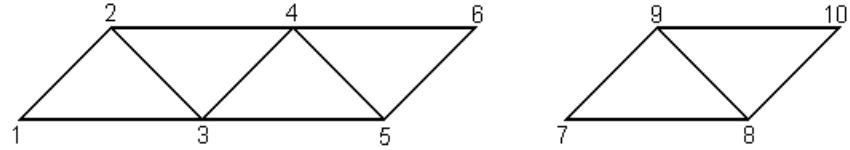
- *Public Function AddTriangleGraphics() As TriangleGraphics*
- 個々の三角形

座標データ

(0,0,0)-(2,0,0)-(1,1,0)  
(2,0,0)-(1,1,0)-(3,1,0)  
(2,0,0)-(4,0,0)-(3,1,0)  
(4,0,0)-(5,1,0)-(3,1,0)

# クライアントグラフィクス

## TriangleStripGraphics



- *Public Function AddTriangleStripGraphics() As \_  
TriangleStripGraphics*
- 接続された三角形

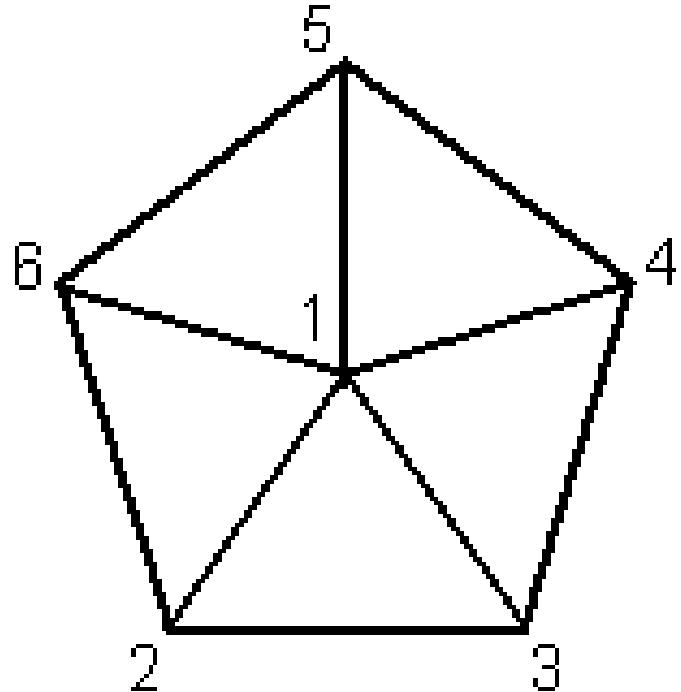
座標データ

(0,0,0), (1,1,0), (2,0,0),  
(3,1,0), (4,0,0), (5,1,0),  
(6,0,0), (8,0,0), (7,1,0),  
(9,1,0)

- 分割長さ  
6, 4

# クライアントグラフィクス

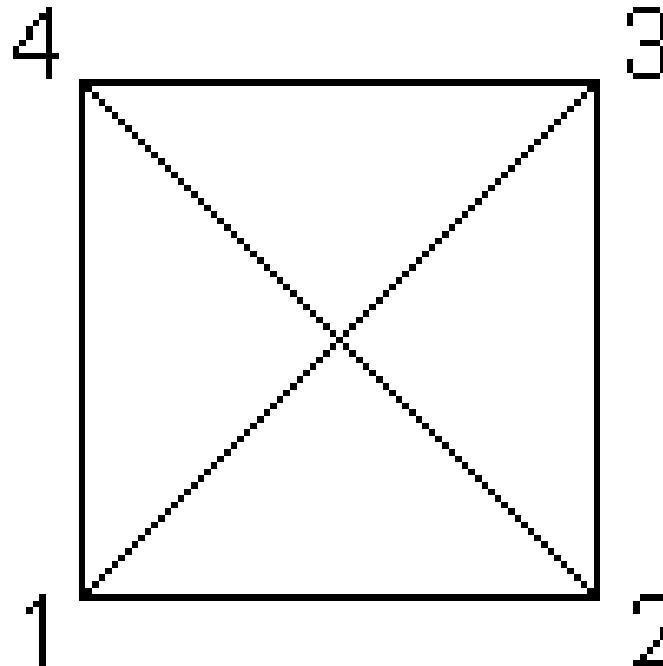
## TriangleFanGraphics



- *Public Function AddTriangleFanGraphics() As TriangleFanGraphics*
- 接続された三角形
  - 座標データ  
(2,3,0), (0,0,0), (4,0,0),  
(5,4,0), (2,6,0), (-1,4,0)
- 分割長さをサポート

# クライアントグラフィクス

## Coordinate Sets の使用



- *GraphicsIndexSet* オブジェクトで *GraphicsCoordinateSet* に追加された点群の関係を定義
- 必要となる点データの最適化にも利用可能。同一の2点は、同じデータを参照可能

座標セット

$(0,0,0), (1,0,0), (1,1,0), (0,1,0)$

*LineGraphics* に指定するインデックス

$(1, 2, 2, 3, 3, 4, 4, 1, 1, 3, 2, 4)$

*LineStripGraphics* に指定するインデックス

$(1, 2, 3, 4, 1, 3, 2, 4)$

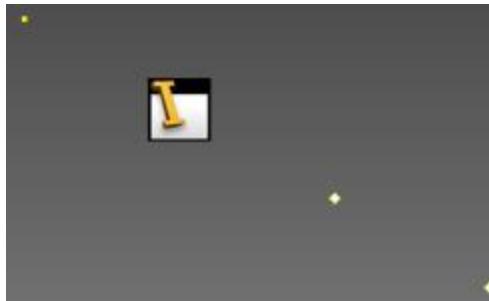
分割長さ (6, 2)

# クライアントグラフィクス

## PointGraphics

- ✖ Point Style
- ✖ On Curve Point Style
- No Snap Point Style
- ♦ Filled Cross Point Style
- Filled Circle Select Point Style
- Filled Circle Point Style
- End Point Style
- ◊ Dim Circular Point Style
- ◊ Cross Point Style
- Circle Point Style

- *AddPointGraphics*
- 点はそれぞれ定義された座標で表示
- 表示スタイルは定義済みのタイプの一つを使用可能
- ビットマップも指定可能



# クライアントグラフィクス

## *TextGraphics*



- *Public Function AddTextGraphics() As TextGraphics*
- 主なプロパティ
  - ✓ Text As String
  - ✓ Anchor As Point
  - ✓ Bold As Boolean
  - ✓ Font As String
  - ✓ FontSize As Integer
  - ✓ HorizontalAlignment As HorizontalTextAlignmentEnum
  - ✓ Italic As Boolean

# クライアントグラフィクス

## Curve Graphics

- *Public Function AddCurveGraphics (ByVal Curve As Object ) As CurveGraphics*
- 入力の*Curve*オブジェクトは以下を指定可能
  - ✓ *LineSegment*
  - ✓ *Circle*
  - ✓ *Arc3d*
  - ✓ *EllipseFull*
  - ✓ *EllipticalArc*
  - ✓ *BSplineCurve*

# クライアントグラフィクス

## Surface Graphics

- *Public Function AddSurfaceGraphics( ByVal Surfaces As Object ) As SurfaceGraphics*
- 入力のSurfacesオブジェクトは以下を指定可能
  - ✓ *SurfaceBody*
  - ✓ *Face*
  - ✓ *Faces*
  - ✓ *FaceCollection*
- 様々なボディプリミティブの作成を行う*TransientBrep* メソッドも入力として使用可能

# クライアントグラフィクス

## *Primitives* と *GraphicsData*

GraphicDataSets

GraphicDataSet (a)

GraphicsColorSet

GraphicsCoordinateSet

GraphicsImageSet

GraphicsIndexSet

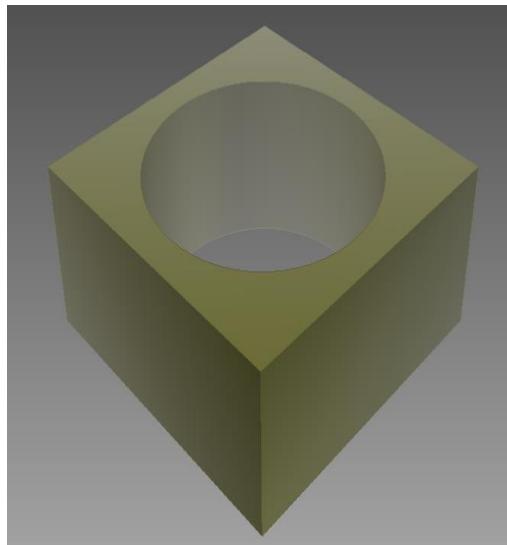
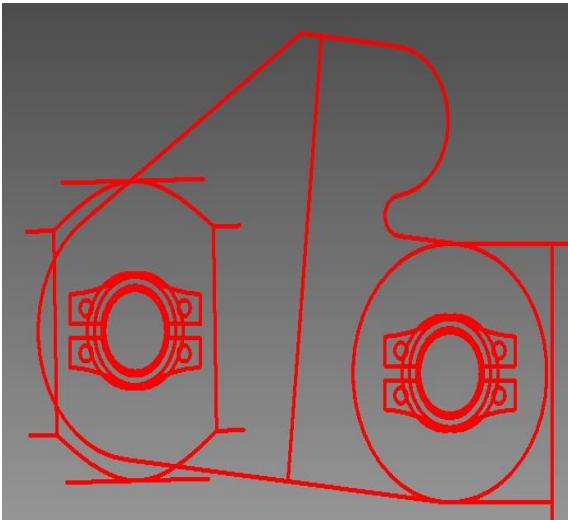
GraphicsNormalSet

GraphicsTextureCoordinateSet

- *GraphicsDataSets* はプリミティブに必要なデータセットの作成が可能
- プリミティブの色、テクスチャまたはイメージのカスタマイズにも用いられる

# クライアントグラフィクス

## ネイティブモデルデータの利用



- *CurveGraphics*
  - ✓ モデルから、ジオメトリのエッジを利用可能。*Edge.Geometry* または*TransientGeometry*で生成したカーブ
- *SurfaceGraphics*
  - ✓ モデルの*SurfaceBody*または*TransientBRep*から生成されたを利用可能
- *ComponentGraphics*
  - ✓ パートまたはアセンブリの*ComponentDefinition*を利用可能

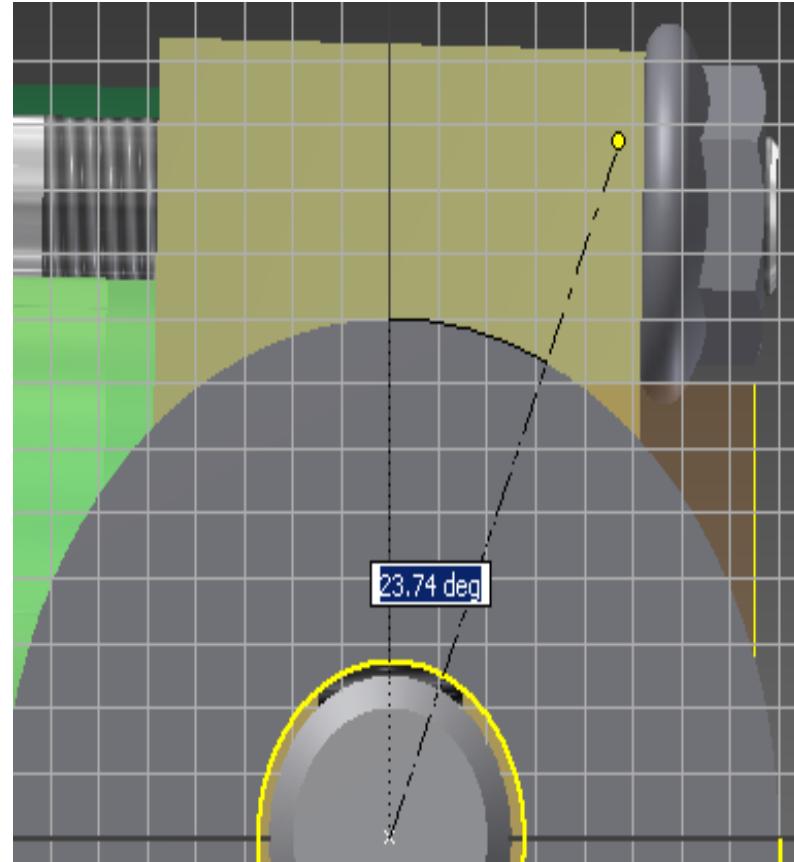
# クライアントグラフィクス

## *InteractionGraphics*オブジェクト

- *InteractionGraphics* は、*InteractionEvents* のコンテキストでのみであるという点を除き *ClientGraphics* と類似した方法で操作が可能
- *InteractionGraphics* は *ClientGraphics* と比較してパフォーマンスに優れ、コマンド中のリアルタイムフィードバックに適している
- *InteractionGraphics* は *InteractionEvents* オブジェクト停止時に自動的に削除される
- *InteractionEvents.InteractionGraphics* プロパティからアクセス

# クライアントグラフィクス

## *Overlay*と*Preview Graphics*



- *InteractionGraphics* は *Preview* または *Overlay* グラフィクスを使用可能
  - *Preview* グラフィクス通常の *ClientGraphics*と同じ
  - *Overlay* グラフィクスは特別な «overlay» 面に独立して描かれる
  - 利用法
    - ✓ Inventorのモデルビューが静止状態で、グラフィクスの更新のみが必要な場合、*Overlay*グラフィクスはより高速に動作
- 例
- スケッチ環境での結果のプレビュー



参照キー



# 参照キー

## 参照キー概要

- 参照キーは、Inventorのオブジェクトから取得可能な永続的なIDであり、後に同じオブジェクトを取得するのに使用される
- 参照キーの使用法は、取得及び参照キーからエンティティ取得するとの両方で、参照キー自体は読み取り専用操作
- 参照キーをサポートするオブジェクトは*GetReferenceKey* メソッドを持つ
- Inventorとアペレンテスの双方でサポートされる

# 参照キー

## オブジェクト参照と参照キー

- 典型的なオブジェクト参照は、限定的な存続期間を持つ

- ✓ Set oFace = oBody.Faces.Item(1)
- ✓ オブジェクトがB-Repp、SurfaceBody, FaceShell, Face, EdgeLoops, Edge, EdgeUse, Vertexオブジェクトであれば、モデル再計算時に参照は失われる
- ✓ また、ドキュメントが閉じられる時に、参照は失われる

- オブジェクト参照は、その時点でのオブジェクトのメソッドやプロパティへのアクセスを提供

# 参照キー

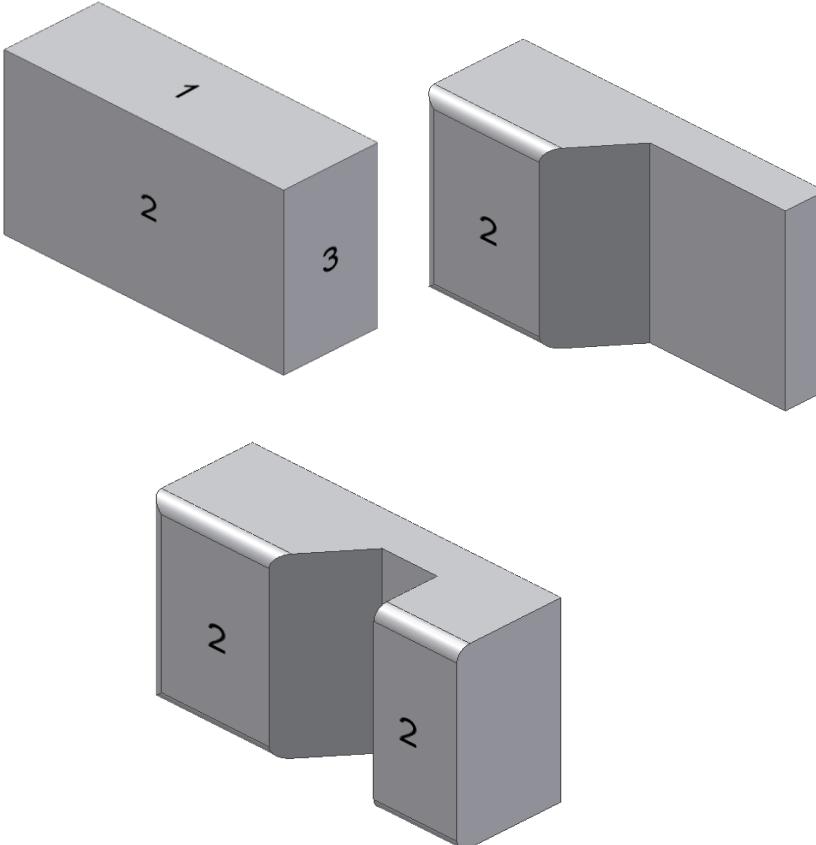
## オブジェクト参照と参照キー

- 参照キーには存続期間の制限はない
  - ✓ セッション間で持続
  - ✓ 編集間で持続
- 参照キーは、オブジェクトの *GetReferenceKey* メソッドにより、取得
- 参照キーはバイト配列であり、エンティティを取得するためだけに利用される。オブジェクトの比較をする場合は参照キーでの比較ではなく、参照キーを使用して取得したオブジェクトの比較を行う
- *ReferenceKeyManager* オブジェクトは、リファレンスキーをオブジェクト参照に戻す機能を提供
  - ✓ *ReferenceKeyManager.BindKeyToObject*



# 参照キー

## B-Repオブジェクトと参照キー



- B-Rep オブジェクトは、モデル作成中および編集中に変更されて行くため、参照キーをサポートすることは非常に困難
- モデル操作により、B-Repを分割することも可能

# 参照キー

## B-Repオブジェクトと参照キー

- 内部では、B-Rep オブジェクトのための参照キーは、オブジェクトへのマップとオブジェクトについての手がかりを定義。この為に、B-Rep 参照キーは、サイズが非常に大きくなる
- 参照キーデータのサイズを最小限にするため、テーブルが作成される。B-Rep 参照キーは、実際はこのテーブルへのポインターであり、このテーブルは、“reference key context”と言われる
- 参照キーを作成または、参照キーからエンティティを取得する時、参照キーコンテキストを提供するひつようがある
- このため、B-Rep 参照キーを保存する時は、参照キーコンテキストも保存しなければならない

# 参照キー

## 参照キー サンプルコード(VB.Net)

```
'Faceまたはドキュメント(パートまたはアセンブリ)の参照キーとコンテキストを取得
Sub GetFaceReferenceKey (ByVal oDoc As Document, ByVal oFace As Face, ByRef oRefKey() As Byte, ByRef oContextData() As Byte)
    ' キーコンテキストを作成 (B-Repの参照キーのために必要)
    Dim KeyContext As Long = oDoc.ReferenceKeyManager.CreateKeyContext()

    ' Faceの参照キーを取得
    oFace.GetReferenceKey(oRefKey, KeyContext)

    ' キーコンテキストを保存
    oDoc.ReferenceKeyManager.SaveContextToArray(KeyContext, oContextData)
End Sub

'参照キーおよびコンテキストからFaceを取得
Function GetFaceFromReferenceKey(ByVal oDoc As Document, ByVal oRefKey() As Byte, ByVal oContextData() As Byte) As Face
    Try
        'バイト配列からコンテキストキーを取得
        Dim oKeyContext As Long = oDoc.ReferenceKeyManager.LoadContextFromArray(oContextData)

        '参照キーをFaceオブジェクトにバインド
        Dim MatchType As Object
        Dim obj As Object = oDoc.ReferenceKeyManager.BindKeyToObject(oRefKey, oKeyContext, MatchType)

        If (TypeOf (obj) Is Face) Then Return obj

        Return Nothing
    Catch ex As Exception
        Return Nothing
    End Try
End Function
```



トランザクション

# トランザクション

## トランザクション

- トランザクションは作成、編集、削除操作をユーザによるUndo・Redoが可能な単位としてカプセル化
- Inventorのデータを変更する全てのコマンドはトランザクション化
- 編集操作に失敗して、トランザクションを完了できなかった場合は、変更されたドキュメントは元の状態に戻される
- トランザクションはフォーマルなデータベーストランザクションとして取り扱われる

# トランザクション

## トランザクション

- ドキュメントの内容を変更する命令が実行されるときは、いつでもトランザクションは発生
- Undoを実行した場合、トランザクションがUndoされる
- 全ての作成・編集APIの実行はトランザクションが発生
- APIでは、トランザクショングループの作成が可能であり、何がUndoされるかを制御が可能
- トランザクションはアプリケーションベースであり、ドキュメントには関連しない
- ドキュメントのクローズにより、トランザクションはクリアーされる
- アドインまたは、スタンドアロンexeで使用可能

# トランザクション

## トランザクションスタック

- トランザクションスタックはアプリケーションスコープを持ち、Inventorないで実行されたすべてのトランザクション化可能なアクションは、アクションがどのドキュメントに対して実行されたかにかかわらず一つのトランザクションリストに保存される
- スタックはドキュメントがクローズされる際にクリアーされる

# トランザクション

## トランザクション サンプル(VB.Net)

```
Public Sub UsingTrans()
    ' アクティブなドキュメントの参照を取得
    Dim oDoc As Document
    oDoc = _InvApplication.ActiveDocument
    Dim oCmpDef As PartComponentDefinition
    oCmpDef = oDoc.ComponentDefinition
    Dim oSketch As PlanarSketch
    oSketch = oCmpDef.Sketches(1)

    Dim oTG As TransientGeometry
    oTG = _InvApplication.TransientGeometry
    ' TransactionManagerを取得
    Dim oTxnMgr As TransactionManager
    oTxnMgr = _InvApplication.TransactionManager
    ' トランザクションの開始
    Dim oTxn As Transaction
    oTxn = oTxnMgr.StartTransaction(oDoc,
                                    "My Rectangle Command")
    ' スケッチラインを追加
    Dim oLine As SketchLine
    oLine = oSketch.SketchLines.AddByTwoPoints(
        oTG.CreatePoint2d(0, 0), oTG.CreatePoint2d(1, 0))
    oLine = oSketch.SketchLines.AddByTwoPoints(
        oLine.EndSketchPoint, oTG.CreatePoint2d(1, 2))
    oLine = oSketch.SketchLines.AddByTwoPoints(
        oLine.EndSketchPoint, oTG.CreatePoint2d(0, 2))
    oLine = oSketch.SketchLines.AddByTwoPoints(
        oLine.EndSketchPoint, oTG.CreatePoint2d(0, 0))
    oTxn.End()
End Sub
```

- *TransactionManager*
  - ✓ トランザクションに関する全ての操作をカプセル化
- *Transaction*
  - 一つのトランザクションを表す

# トランザクション

## トランザクション サンプル(VB.Net)

```
Public Sub start_end_abort()
    ' アクティブドキュメントの参照を取得
    Dim oDoc As Document
    oDoc = _InvApplication.ActiveDocument

    ' TransactionManagerを取得
    Dim oTxnMgr As TransactionManager
    oTxnMgr = _InvApplication.TransactionManager

    ' トランザクションの開始
    Dim oTxn As Transaction
    oTxn = oTxnMgr.StartTransaction(oDoc, "MyTransaction")

    Try
        ' オペレーションを実行・・・

        ' トランザクションを終了
        oTxn.End()

    Catch ex As Exception
        ' オペレーション中にエラーが発生した場合
        ' トランザクションをアボートし変更を破棄
        MsgBox("オペレーション中にエラー発生")
        oTxn.Abort()

    End Try

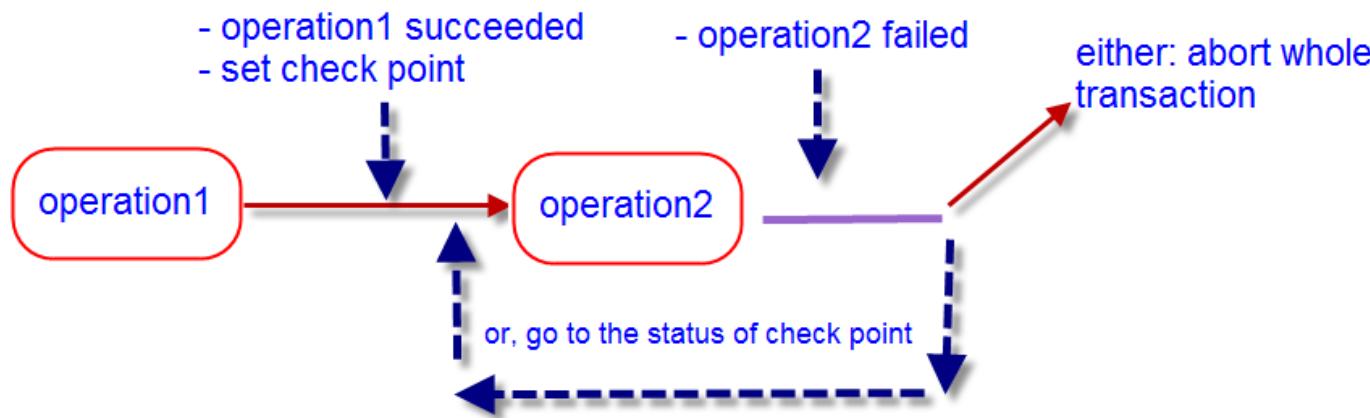
End Sub
```

- 各トランザクションの開始は、終了またはアボートとペアになっていること
- エラーが発生した場合、トランザクションをアボートする

# トランザクション

## チェックポイント

- トランザクション内のロールバックポイントへのブックマーク
- Inventorはチェックポイントを子トランザクションとして実装
- *GoToCheckpoint* は本質的にはチェックポイントトランザクションの破棄



# トランザクション

## チェックポイント サンプル(VB.net)

```
' アクティブドキュメントの参照を取得
Dim oDoc As Document
oDoc = ThisApplication.ActiveDocument

' TransactionManagerの取得
Dim oTxnMgr as TransactionManager
oTxnMgr = ThisApplication.TransactionManager

' トランザクション開始
Dim oTxn1 As Transaction
oTxn1 = oTxnMgr.StartTransaction(oDoc, "Checkpoint Txn")

' *****
' 押し出しの作成
' *****

' 押し出し作成の前にチェックポイントを作成
Dim oChkPt as CheckPoint
oChkPt = oTxnMgr.SetCheckPoint

' *****
' 押し出しを作成
' *****

' エラー発生時にチェックポイントに戻す
If Err Then
    MsgBox "押し出し作成に失敗しました"
    oTxnMgr.GoToCheckPoint oChkPt
End If
```

# トランザクション

## トランザクション イベント (VB.Net)

- トランザクションに関連するイベント通知の受け取りが可能
- トランザクションのCommit、undo、redo、abortおよびdeleteをイベント通知

```
Dim oTransMgr As TransactionManager
oTransMgr = _InvApplication.TransactionManager

Dim oTransEvents As TransactionEvents
oTransEvents = oTransMgr.TransactionEvents

Public Sub startEvents()
    'delegate the event of undo
    AddHandler oTransEvents.OnUndo, AddressOf OnUndo
End Sub

Public Sub stopEvents()
    'remove the event of undo
    RemoveHandler oTransEvents.OnUndo, AddressOf OnUndo
End Sub
```

# トランザクション

## 入れ子トランザクション

- *TransactionManager::StartTransaction* メソッドは親または子トランザクションの開始となる
- 別のトランザクションスコープ内のトランザクションの開始は子トランザクションの開始となる

# トランザクション

## トランザクション利用時の注意事項

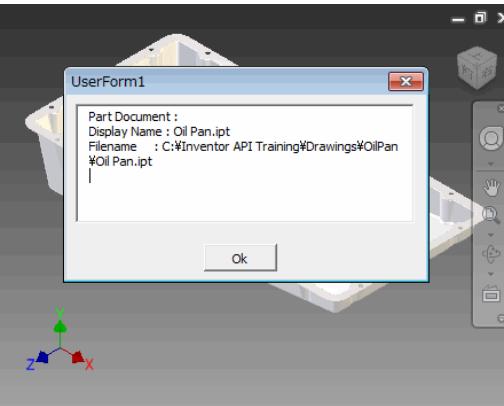
- UndoまたはRedoは実行中のトランザクションがない場合にのみ実行する
- Inventorのコマンドをトランザクション内に組み込まない
- トランザクション内に、ユーザインタラクションイベントを組み込まない。モーダルダイアログの作成はOKだが、モードレスダイアログは作成しない
- トランザクションイベントハンドラ内で、トランザクションに関連する操作を行わない。例としてコミットトランザクションイベントハンドラ内で、スケッチラインを作成することは不正な操作となる
- トランザクション中にドキュメントを閉じないこと。例外として、同一トランザクション内で開かれたドキュメントを閉じることは許容される

# 演習の解答

# 演習1

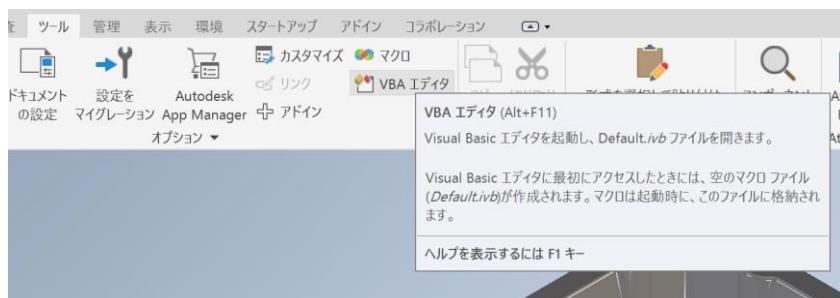
## VBA

VBA マクロを作成します。テキスト ボックスに、アクティブなドキュメントに関する情報としてファイル名や表示名を表示します

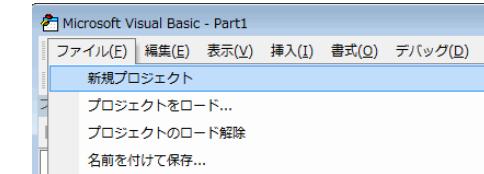


1. Inventor を起動して、C:\Inventor API Training\Drawings\OilPan\OilPan.ipt を開きます。

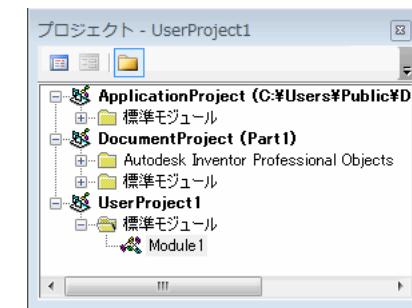
2. VBA エディタを起動します



3. 新しいプロジェクトを作成します



4. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習1

## VBA

5. メインの編集ウィンドウで、アクティブなドキュメントのタイプ(ActiveDocumentType)、およびアクティブなドキュメント(ActiveDocument)の表示名(DisplayName)、ファイル名(FullFileName)を取得するコードを記述します。初めにThisApplication をセットするのを忘れないようにしてください

```
Option Explicit
Dim oApp As Inventor.Application

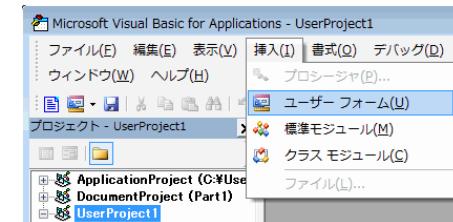
Public Sub InfoDoc()
    Dim oDoc As Document
    Set oApp = ThisApplication

    Dim sPrompt As String
    Set oDoc = oApp.ActiveDocument
    '実際にドキュメントがオープンされているかどうかをチェックします。
    If oDoc Is Nothing Then
        MsgBox "There is no active document"
        Exit Sub
    End If

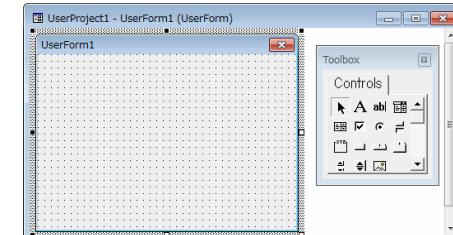
    Select Case oApp.ActiveDocumentType
        Case kAssemblyDocumentObject
            sPrompt = "Assembly Document :" & vbCrLf
        Case kPartDocumentObject
            sPrompt = "Part Document :" & vbCrLf
        Case kUnknownDocumentObject
            sPrompt = "Unknown Document :" & vbCrLf
    End Select
End Sub
```

```
sPrompt = sPrompt & "Display Name : " & oDoc.DisplayName & vbCrLf
sPrompt = sPrompt & "Filename : " & oDoc.FullName & vbCrLf
UserForm1.TextBox1.Text = sPrompt
UserForm1.Show vbModal
End Sub
```

6. ドキュメントの情報を小さなダイアログに表示させるため、新しいフォームを作成します。メニューから[挿入]、[ユーザー フォーム]の順に選択します



7. テキスト ボックスとコマンド ボタンを 1 つずつフォームに追加します。テキスト ボックス の MultiLine プロパティを True にします



# 演習1

## VBA

8. UserForm1 フォームに追加したボタンのクリックプロシージャに、  
Unload Me とコードを記入します

9. VBA マクロを起動するには、IDE を使用するか、Inventor でメ  
ニューから [ツール]、[マクロ] の順に選択します

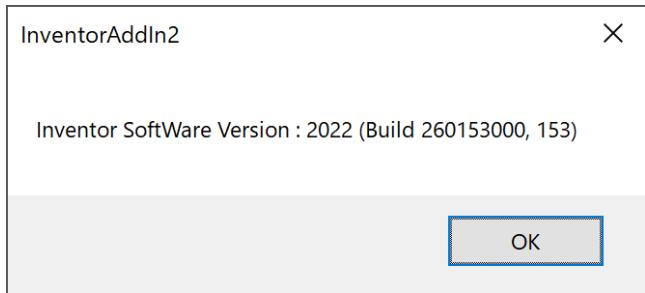


演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

# 演習2

## Inventor アドイン

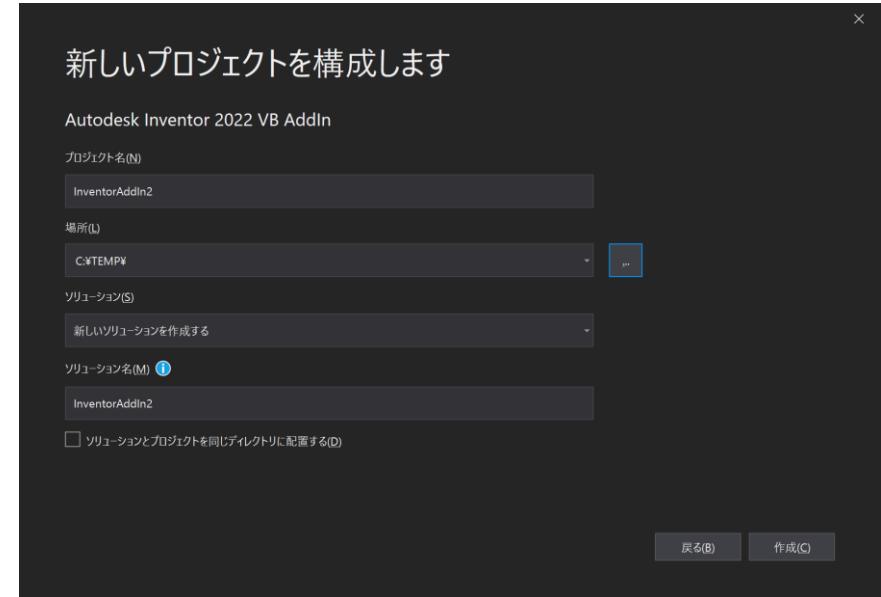
Inventorがスタートした時に、Inventorのカレントのバージョンを見せるメッセージボックスを表示するアドインを作成します。



1. Visual Studio 2019 を起動して、新規作成で、マイテンプレート内で”Autodesk Inventor 2022 VB AddIn”を選択し Wizardによるアドインを作成します。



2. アドイン作成用の、プロジェクト名を記述します



# 演習2

## Inventor アドイン

3. プロジェクト名は、アドイン作成時の”プロジェクト名”になり、クラス名は自動的に“StandardAddInServer”が割り当てられます。

StandardAddInServer.vbを開き、Activateコールバック関数に  
MsgBoxステートメントを追加します。

```
Public Sub Activate(ByVal addInSiteObject As Inventor.ApplicationAddInSite, _  
    ByVal firstTime As Boolean) Implements Inventor.ApplicationAddInServer.Activate  
    ' Initialize AddIn members.  
    g_inventorApplication = addInSiteObject.Application  
  
    MsgBox(" Inventor SoftWare Version :" & _  
        g_inventorApplication.SoftwareVersion.DisplayName)  
End Sub
```

4. ソリューションエクスプローラー内から  
Autodesk.InventorAddIn.Inventor.addinファイルを選択し、  
<Assembly>設定記述を、アドインのフルパスに置き換える。

```
<Addin Type="Standard">  
    <!--Created for Autodesk Inventor Version 25.0-->  
    <ClassId>{9364a88f-47f1-45bb-b482-2dec685037b2}</ClassId>  
    <ClientId>{9364a88f-47f1-45bb-b482-2dec685037b2}</ClientId>  
    <DisplayName>InventorAddIn1</DisplayName>  
    <Description>InventorAddIn1</Description>  
    <Assembly>{ここにAddin dllへのフルパスを記載}</Assembly>  
    <LoadOnStartUp>1</LoadOnStartUp>  
    <UserUnloadable>1</UserUnloadable>  
    <Hidden>0</Hidden>  
    <SupportedSoftwareVersionGreater Than>24..</SupportedSoftwareVersionGreater Than>  
    <DataVersion>1</DataVersion>  
    <UserInterfaceVersion>1</UserInterfaceVersion>  
</Addin>
```

5. Inventor 2022バージョンのみ動作するものとし、「Inventor Version に依存」指定場所のホルダーに、  
Autodesk.InventorAddIn1.Inventor.addinファイルを複写する。

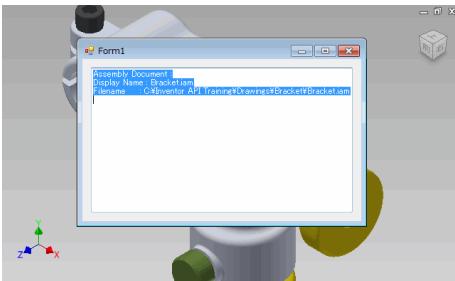
C:\ProgramData\Autodesk\Inventor 2022\Addins\

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

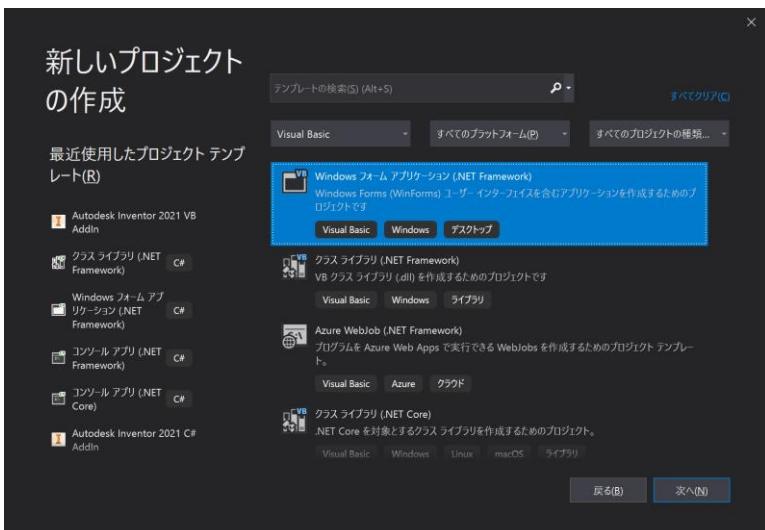
# 演習3

## 外部アプリケーションからの操作

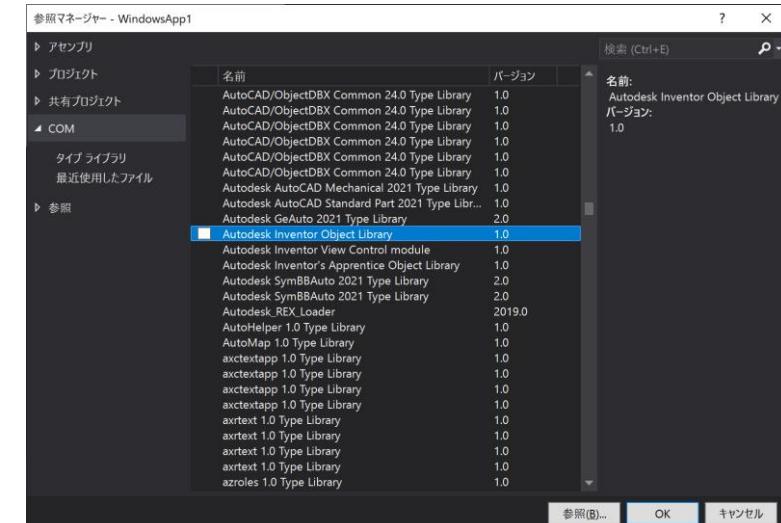
Inventor が実行されていない場合に Inventor を起動する外部アプリケーションを作成します。このアプリケーションはアクティブなドキュメントのプロパティを表示します。



1. Visual Studio 2019を起動して、新しい Windows アプリケーション のプロジェクトを作成します。



2. “プロジェクト”の”参照の追加”からAutodesk Inventor Object Library への参照設定を追加します。



# 演習3

## 外部アプリケーションからの操作

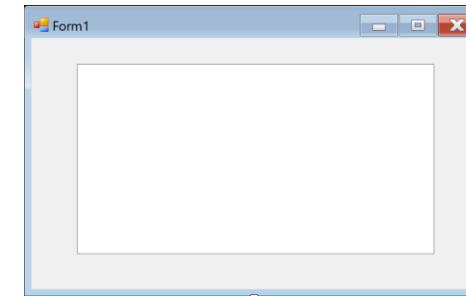
3. このサンプルと演習では Inventor.Application 型の oApp というグローバル変数を使用します。クラス内でInventorオブジェクトを参照できるようにImportsステートメントを使って Inventor.DocumentTypeEnumを記述します。

4. Inventor.Application オブジェクトを作成または取得して、可視状態にします。CreateObject および GetObject については、VB のドキュメントを参照してください。

```
Imports Inventor.DocumentTypeEnum
Public Class Form1
    Public oApp As Inventor.Application
    Private Function InitInventor(Optional ByVal bVisible As Boolean = True) As Boolean
        ' Inventorのインスタンスの獲得。
        Try
            oApp = GetObject(, "Inventor.Application")
        Catch ex As Exception
            ' Inventorのインスタンスの作成。
            Try
                oApp = CreateObject("Inventor.Application")
            Catch exx As Exception
                'インスタンス獲得に失敗。
                MsgBox("Inventorの起動に失敗しました。")
                InitInventor = False
                Exit Function
            End Try
        End Try
    End Function
End Class
```

```
' Inventorを表示。
oApp.Visible = True
InitInventor = True
End Function
End Class
```

5. フォームに テキスト ボックス を追加し、Multiline プロパティを True にします。



AllowDrop	False
CharacterCasing	Normal
ContextMenuStrip	(none)
Enabled	True
HideSelection	True
ImeMode	NoControl
MaxLength	32767
Multiline	True
PasswordChar	
ReadOnly	False
ShortcutsEnabled	True
TabIndex	0
Text	The text associated with the control.

ソリューション エクス... プロパティ チーム エクスプローラー

# 演習3

## 外部アプリケーションからの操作

6. ActiveDocument を取得し、ActiveDocumentType の値に応じたドキュメント タイプ、および表示名(DisplayName)やファイル名(FullFileName)を表示します。

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim oDoc As Inventor.Document
    If InitInventor(True) = False Then
        MsgBox("Inventorの起動に失敗しました")
        Exit Sub
    End If

    oDoc = oApp.ActiveDocument
    If oDoc Is Nothing Then
        TextBox1.Text = "アクティブなドキュメントがありません"
        Exit Sub
    End If

    Select Case oApp.ActiveDocumentType
        Case kAssemblyDocumentObject
            TextBox1.Text = "Assembly Document :" & vbCrLf
        Case kPartDocumentObject
            TextBox1.Text = "Part Document :" & vbCrLf
        Case kUnknownDocumentObject
            TextBox1.Text = "Unknown Document :" & vbCrLf
    End Select

    TextBox1.Text = TextBox1.Text & "Display Name :" & oDoc.DisplayName & vbCrLf
    TextBox1.Text = TextBox1.Text & "Filename :" & oDoc.FullFileName & vbCrLf
End Sub
```

7. Formが破棄されるタイミングで使用したInventorオブジェクトの破棄をします。

```
Private Sub Form1_Disposed(sender As Object, e As EventArgs) Handles Me.Disposed
    'Inventorのオブジェクトを破棄する
    oApp.Quit()

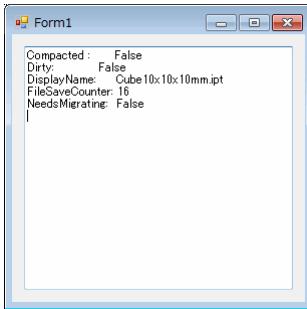
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

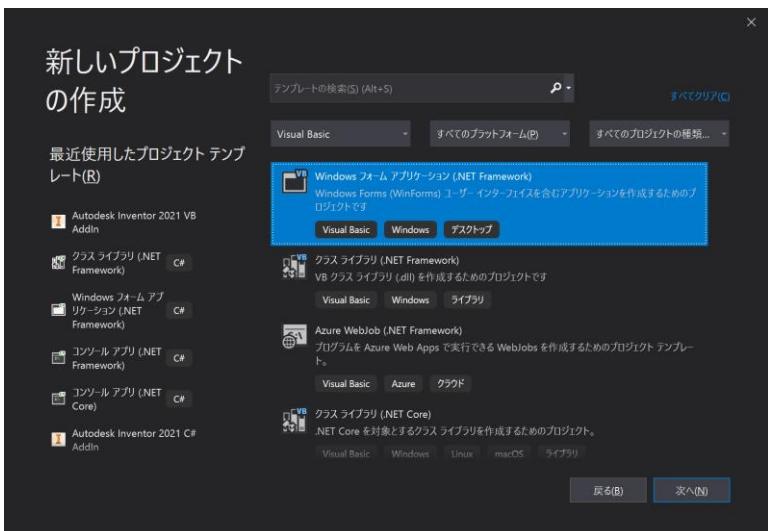
# 演習4

## Apprentice Server

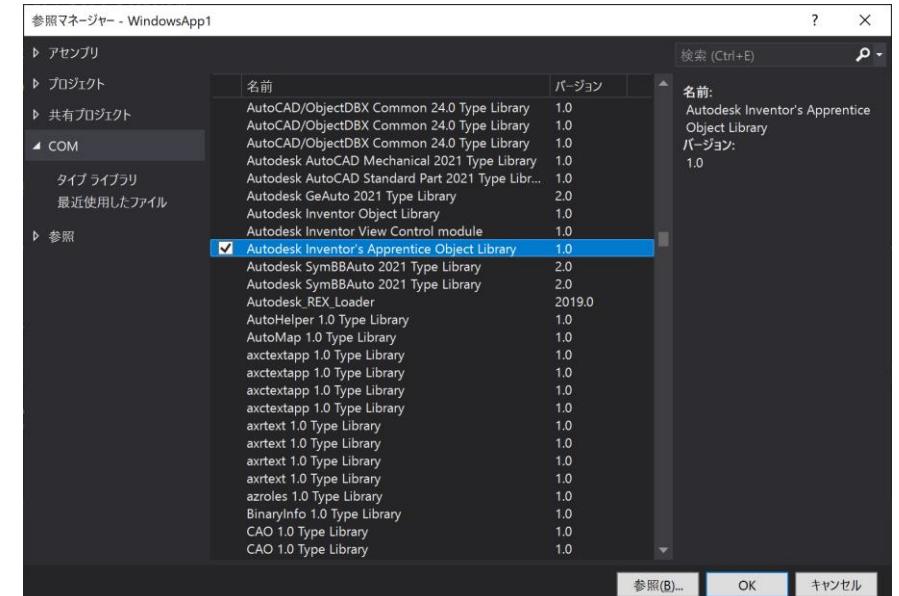
Inventor がインストールされていないPCにおいて Inventor のファイルにアクセスする外部アプリケーションを作成します。



1. Visual Studio 2019を起動して、新しい Windows アプリケーション のプロジェクトを作成します。



2. “COM”タブを開き、Autodesk Inventor's Apprentice Object Library への参照設定を追加します。



# 演習4

## Apprentice Server

3. Imports InventorApprenticeを使用し、グローバル変数 ApprenticeServerComponent を作成します。

4. Form1\_Load 関数で、New ステートメントを使用して ApprenticeServerComponent を初期化します。

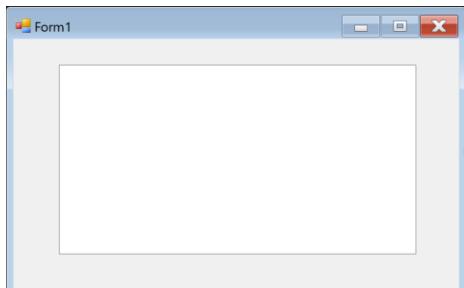
```
Imports InventorApprentice

Public Class Form1
    Public oApp As ApprenticeServerComponent

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
        oApp = New ApprenticeServerComponent
    End Sub

End Class
```

5. フォームにテキスト ボックスを追加し、Multiline プロパティを True にします。



AllowDrop	False
CharacterCasing	Normal
ContextMenuStrip	(none)
Enabled	True
HideSelection	True
ImeMode	NoControl
MaxLength	32767
Multiline	True
PasswordChar	
ReadOnly	False
ShortcutsEnabled	True
TabIndex	0
Text	The text associated with the control.

5. 次に、ApprenticeServerDocument をオープンしてドキュメント情報を表示するコードを追加します。

Imports InventorApprentice

```
Public Class Form1
    Public oApp As ApprenticeServerComponent

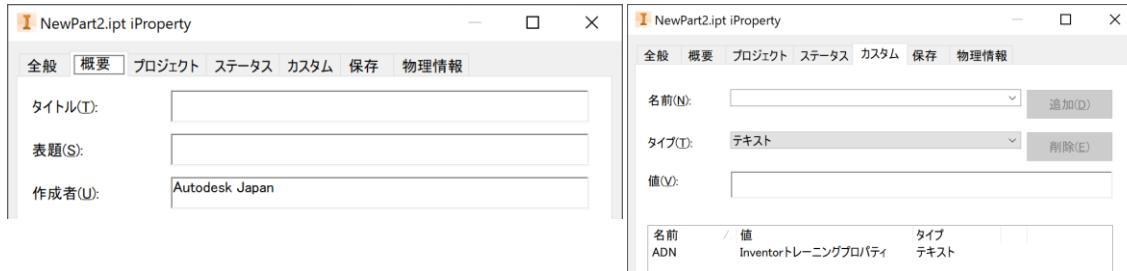
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
        oApp = New ApprenticeServerComponent
        Dim oDoc As ApprenticeServerDocument
        oDoc = oApp.Open( "C:\Inventor API Training\Drawings\Cube10x10x10mm.ipt" )
        With TextBox1
            .Text = ""
            .Text = .Text & "Compacted : " & oDoc.Compacted & vbCrLf
            .Text = .Text & "Dirty: " & oDoc.Dirty & vbCrLf
            .Text = .Text & "DisplayName: " & oDoc.DisplayName & vbCrLf
            .Text = .Text & "FileSaveCounter: " & oDoc.FileSaveCounter & vbCrLf
            .Text = .Text & "NeedsMigrating: " & oDoc.NeedsMigrating & vbCrLf
        End With
    End Sub
End Class
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

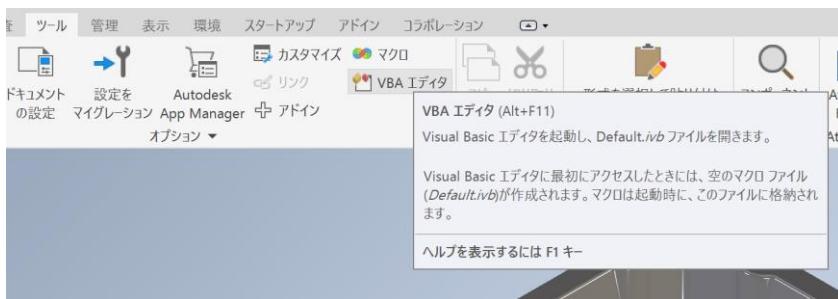
# 演習5

## iProperty の作成と保存

新規にパートドキュメントを作成し、“Inventor 概要情報”にある“作成者”プロパティの値を変更し、カスタムタブ内に、“ADN”というカスタムプロパティを作成し、値を“Inventorトレーニングプロパティ”という文字をセットし、C:\Temp\NewPart2.ipt としてドキュメントを保存して、ドキュメントをクローズする



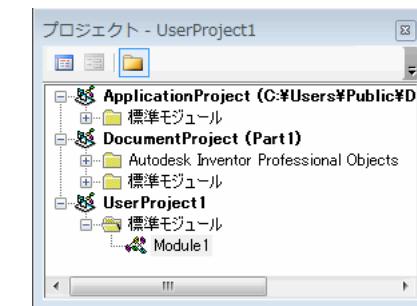
### 1. VBA エディタを起動します



### 2. 新しいプロジェクトを作成します



### 3. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習5

## iProperty の作成と保存

4. 新規パートドキュメントを作成し、PropertySets.Itemで表示名 "Inventor 概要情報" 内部名 "{ F29F85E0-4FF9-1068-AB91-08002B27B3D9 }" を用いてPropertySetを取り出し、ItemByPropIdでPropertyの“作成者プロパティ”的値を編集する。

```
'新規パートドキュメントの作成.  
Dim oDoc As PartDocument  
Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject, _  
    ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))  
  
'あなたの名前を含む為に、“作成者プロパティ”的値を編集。  
'表示名“Inventor 概要情報”  
'内部名称{F29F85E0-4FF9-1068-AB91-08002B27B3D9}  
Dim oSummaryInfoPropSet As PropertySet  
Set oSummaryInfoPropSet = oDoc.PropertySets.Item("{F29F85E0-4FF9-1068-  
    AB91-08002B27B3D9}")  
  
Dim oAuthorProp As Property  
Set oAuthorProp =  
    oSummaryInfoPropSet.ItemByPropId(kAuthorSummaryInformation)  
  
oAuthorProp.Value = "Autodesk Japan"
```

5. 内部名 "{D5CDD505-2E9C-101B-9397-08002B2CF9AE}" を用いてPropertySetを取り出し、Addメソッドでカスタムタブ内に、“ADN”というカスタムプロパティを作成し 値を“Inventorトレーニングプロパティ”という文字をセットし、C:\Temp\NewPart2.ipt としてドキュメントを保存した後、ドキュメントをクローズする

```
'値が“Inventorトレーニングプロパティ”で“ADN”と呼ばれる、  
'新しいカスタムプロパティ(ユーザー定義)の作成。  
'内部名称{D5CDD505-2E9C-101B-9397-08002B2CF9AE}  
Dim oCustomPropSet As PropertySet  
Set oCustomPropSet = oDoc.PropertySets.Item("{D5CDD505-2E9C-101B-  
    9397-08002B2CF9AE}")  
  
Dim oSupplierProp As Property  
Set oSupplierProp = oCustomPropSet.Add("Inventorトレーニングプロパティ",  
    "ADN")  
  
"“C:\Temp\NewPart.ipt” としてドキュメントの保存。  
Call oDoc.SaveAs("C:\Temp\NewPart2.ipt", False)  
  
'ドキュメントのクローズ  
oDoc.Close      '(True or False)
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

# 演習6

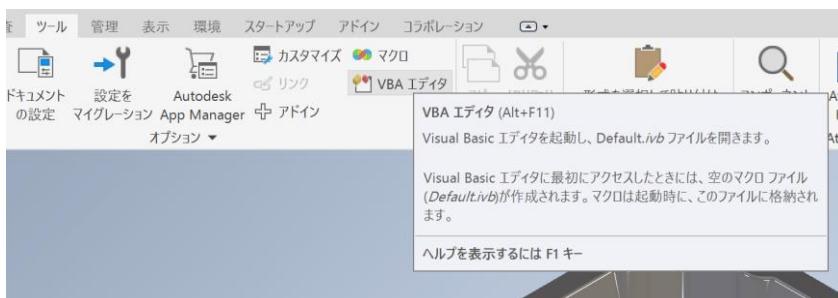
## ParameterとUOM

既存のドキュメントのパラメータより、Length の名称を持つパラメータの値を モードレスダイアログ から変更します。テキストフィールドは、無効な数値が入力されたり定義された場合、赤色の表示をさせます。“更新”ボタンが押されたとき、パラメータに割り当て、モデルを更新します。



1. Inventor を起動して、C:\Inventor API Training\Drawings\Extrude.ipt”を開きます。

2. VBA エディタを起動します



3. 新しいプロジェクトを作成します



4. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習6

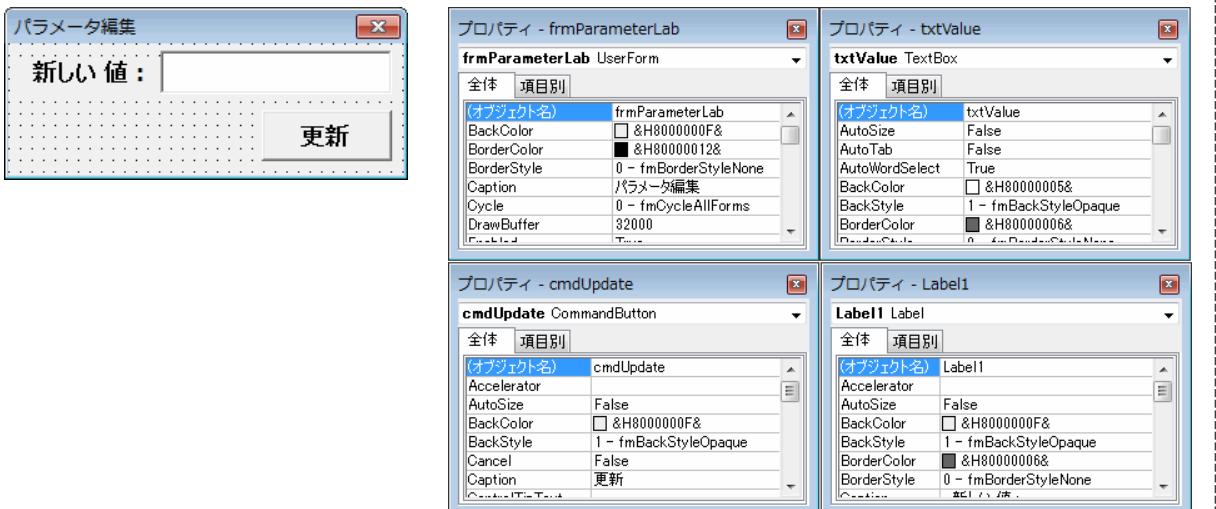
## ParameterとUOM

5. 挿入 -> フォームでユーザーフォームを追加し、プロパティウィンドウで、オブジェクト名を frmParameterLab とし、Caption を“パラメータ編集”とする。

テキストボックスを配置し、オブジェクト名を txtValue とする。

コマンドボタンを配置し、オブジェクト名を cmdUpdate とし、Captionを“更新”とする。

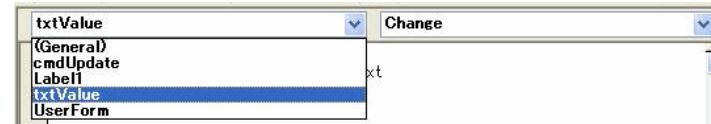
ラベルを配置し、Captionを“新しい値：“とする。



6. ParameterLabマクロを作成し、モードレスで表示する。

```
Public Sub ParameterLab()
    frmParameterLab.Show vbModeless
End Sub
```

7. txtValueオブジェクトの Changeプロシジャーを実装し、  
UnitsOfMeasureのCompatibleUnitsを用いて入力が妥当かの  
判断をし、NGの場合はテキストを赤色にする。



```
Private Sub txtValue_Change()
    Dim oUOM As UnitsOfMeasure
    Set oUOM = ThisApplication.ActiveDocument.UnitsOfMeasure
    On Error Resume Next
```

' 表現が妥当な長さかチェック.

```
Dim bCompatible As Boolean
bCompatible = oUOM.CompatibleUnits(txtValue.Text, _
    kDefaultDisplayLengthUnits, "1", _
    kDefaultDisplayLengthUnits)
```

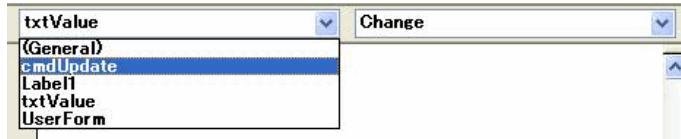
' 成功かを判定し、テキストを適当な意図に変更.

```
If Err Or Not bCompatible Then
    txtValue.ForeColor = vbRed
Else
    txtValue.ForeColor = vbWindowText
End If
End Sub
```

# 演習6

## ParameterとUOM

8. cmdUpdateオブジェクトの Clickプロシジャーを実装し、テキストボックス内の判定をした後、パートドキュメント内のParameterより“Length”パラメータを獲得し、Expressionメソッドにてテキスト入力値で変更し、Updateでドキュメントを更新する。



```
Private Sub cmdUpdate_Click()
    'テキストボックス内が、赤色は未入力ならばエラー
    If txtValue.ForeColor = vbRed Or txtValue.Text = "" Then
        MsgBox "無効な値が入力されました."
        Exit Sub
    End If

    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.ActiveDocument

    '"Length"を持つパラメータを得る。
    On Error Resume Next

    Dim oParam As Parameter
    Set oParam = oDoc.ComponentDefinition.Parameters.Item("Length")

    If Err Then
        MsgBox "パラメータに ""Length"" が見つかりません"
        Exit Sub
    End If
```

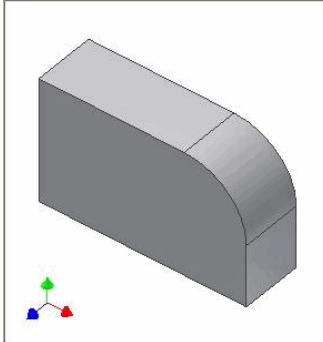
```
' パラメータを変更
oParam.Expression = txtValue.Text
' ドキュメントを更新
oDoc.Update
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

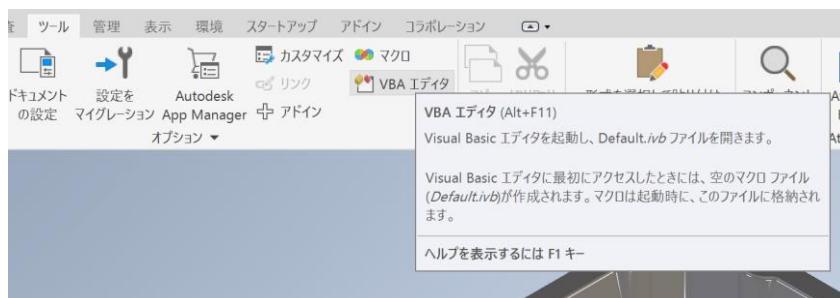
# 演習7

## スケッチとフィーチャーの作成

新規パートドキュメントを作成してスケッチ上にX-Y平面を作成し、  
 PARTS の外形線を作成し、押し出しフィーチャーを作成する。作成  
 したプログラムは”演習8”で使用しますので、任意の名前を付けて  
 保存しておいてください。



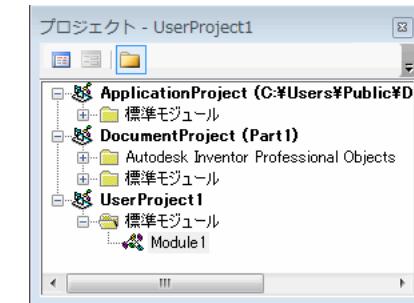
### 1. VBA エディタを起動します



### 2. 新しいプロジェクトを作成します



### 3. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習7

## スケッチとフィーチャーの作成

4. テンプレートを用いてパートファイルを新規に作成し、トランザクションを開始します。

```
Public Sub CreateFeature()  
  
    ' パーツファイルの新規作成  
    Dim oDoc As PartDocument  
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject, _  
        ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))  
  
    Set oDoc = ThisApplication.ActiveDocument  
  
    ' トランザクション開始  
    Dim oTrans As Transaction  
    Set oTrans = ThisApplication.TransactionManager.StartTransaction(oDoc, _  
        "Feature Lab")
```

5. ベース側のPlanarスケッチを新規作成し、ジオメトリの獲得した後にベース側スケッチポイントを設定します。

```
'ベース側のPlanarスケッチの新規作成  
Dim oSketch As PlanarSketch  
Set oSketch = oDoc.ComponentDefinition.Sketches.Add(_  
    oDoc.ComponentDefinition.WorkPlanes.Item(3))  
  
'ジオメトリの獲得  
Dim oTG As TransientGeometry  
Set oTG = ThisApplication.TransientGeometry  
  
'ベース側スケッチポイントを設定  
Dim oPoints(5) As SketchPoint  
Set oPoints(0) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 0), False)  
Set oPoints(1) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(10, 0), False)  
Set oPoints(2) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(10, 3), False)  
Set oPoints(3) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(7, 6), False)  
Set oPoints(4) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(7, 3), False)  
Set oPoints(5) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 6), False)
```

# 演習7

## スケッチとフィーチャーの作成

6. スケッチポイントを使ってLINEと円弧を作成しベース側となるスケッチ外周形状を作成します。

```
'ベース側スケッチ外周を作成
'スケッチポイントを使ってLINEを作成
Dim oLines(3) As SketchLine
Dim oArc As SketchArc

Set oLines(0) = oSketch.SketchLines.AddByTwoPoints(oPoints(0), oPoints(1))
Set oLines(1) = oSketch.SketchLines.AddByTwoPoints(oPoints(1), oPoints(2))
Set oLines(2) = oSketch.SketchLines.AddByTwoPoints(oPoints(3), oPoints(5))
Set oLines(3) = oSketch.SketchLines.AddByTwoPoints(oPoints(5), oPoints(0))

'Sケッチポイントを使って円弧を作成
Set oArc = oSketch.SketchArcs.AddByCenterStartEndPoint(oPoints(4), _
oPoints(2), oPoints(3))
```

7. 外周形状を使ってベース側のプロファイルを作成し、押し出しフィーチャーを作成し、トランザクションを終了します。

```
'ベース側のプロファイルの作成
Dim oProfile As Profile
Set oProfile = oSketch.Profiles.AddForSolid

'プロファイルを使ってベース側の押し出しフィーチャーの新規作成
Dim oBaseExtrude As ExtrudeFeature
Set oBaseExtrude = _
oDoc.ComponentDefinition.Features.ExtrudeFeatures.AddByDistanceExtent(_
oProfile, 3, kPositiveExtentDirection, kJoinOperation)

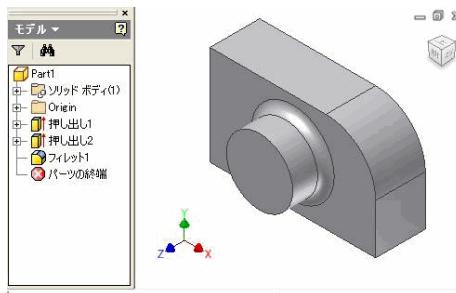
'Tランザクションの終了
oTrans.End
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

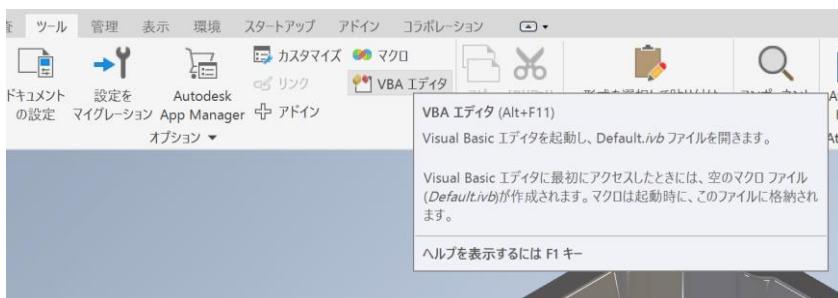
# 演習8

## B-Rep によるアクセスとフィレットの作成

演習7で作成した押し出しフィーチャーのエンドフェースを使ってボス側の押し出しフィーチャーを追加作成し、ベース側とボス側間のエッジにフィレットを作成する。



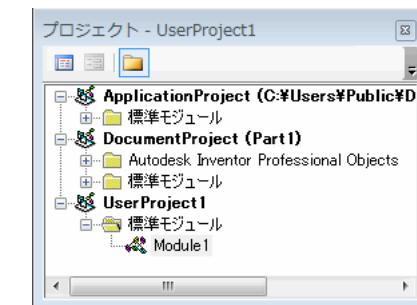
### 1. VBA エディタを起動します



### 2. 新しいプロジェクトを作成します



### 3. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習8

## B-Rep によるアクセスとフィレットの作成

4. 演習7 の処理：新規パートドキュメントを作成してスケッチ上にX-Y平面を作成し、パートの外形線を作成し、押し出しフィーチャーを作成します。

```
Public Sub CreateFeature()
    ' パーツファイルの新規作成
    Dim oDoc As PartDocument
    Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject,
        ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    Set oDoc = ThisApplication.ActiveDocument

    ' トランザクション開始
    Dim oTrans As Transaction
    Set oTrans = ThisApplication.TransactionManager.StartTransaction(oDoc,
        "Feature Lab")
    ' ベース側のPlanarスケッチの新規作成
    Dim oSketch As PlanarSketch
    Set oSketch = oDoc.ComponentDefinition.Sketches.Add(
        oDoc.ComponentDefinition.WorkPlanes.Item(3))

    ' ジオメトリの獲得
    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    ' ベース側スケッチポイントを設定
    Dim oPoints(5) As SketchPoint
    Set oPoints(0) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 0), False)
    Set oPoints(1) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(10, 0), False)
    Set oPoints(2) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(10, 3), False)
    Set oPoints(3) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(7, 6), False)
    Set oPoints(4) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(7, 3), False)
    Set oPoints(5) = oSketch.SketchPoints.Add(oTG.CreatePoint2d(0, 6), False)
```

```
' ベース側スケッチ外周を作成
' スケッチポイントを使ってLINEを作成
Dim oLines(3) As SketchLine
Dim oArc As SketchArc

Set oLines(0) = oSketch.SketchLines.AddByTwoPoints(oPoints(0), oPoints(1))
Set oLines(1) = oSketch.SketchLines.AddByTwoPoints(oPoints(1), oPoints(2))
Set oLines(2) = oSketch.SketchLines.AddByTwoPoints(oPoints(3), oPoints(5))
Set oLines(3) = oSketch.SketchLines.AddByTwoPoints(oPoints(5), oPoints(0))

' スケッチポイントを使って円弧を作成
Set oArc = oSketch.SketchArcs.AddByCenterStartEndPoint(oPoints(4),
    oPoints(2), oPoints(3))
' ベース側のプロファイルの作成
Dim oProfile As Profile
Set oProfile = oSketch.Profiles.AddForSolid

' プロファイルを使ってベース側の押し出しフィーチャーの新規作成
Dim oBaseExtrude As ExtrudeFeature
Set oBaseExtrude = _
    oDoc.ComponentDefinition.Features.ExtrudeFeatures.AddByDistanceExtent(
        oProfile, 3, kPositiveExtentDirection, kJoinOperation)
```

# 演習8

## B-Rep によるアクセスとフィレットの作成

5. ベース側の押し出しフィーチャーのエンドフェースを使ってボス側のスケッチの追加新規作成し、スケッチ上に円を作成した後にボス側のプロファイルを作成します。

プロファイルを使ってボス側の押し出しフィーチャーを新規作成します。

```
' ベース側の押し出しフィーチャーのエンドフェースを使ってボス側のスケッチの追加新規作成.  
Set oSketch =  
oDoc.ComponentDefinition.Sketches.Add(oBaseExtrude.EndFaces.Item(1))  
  
' スケッチ上に円を作成  
Call oSketch.SketchCircles.AddByCenterRadius(oTG.CreatePoint2d(5, 3), 2)  
  
' ボス側のプロファイルの作成  
Set oProfile = oSketch.Profiles.AddForSolid  
  
' プロファイルを使ってボス側の押し出しフィーチャーの新規作成  
Dim oBossExtrude As ExtrudeFeature  
Set oBossExtrude =  
oDoc.ComponentDefinition.Features.ExtrudeFeatures.AddByDistanceExtent(  
oProfile, 2, kPositiveExtentDirection, kJoinOperation)
```

6. ベース側のSideFacesとボス側のEndFacesより間のエッジを見つける。

```
' ベース側とボス側のフェースを取り出す.  
Dim oSideFace As Face  
Set oSideFace = oBossExtrude.SideFaces.Item(1)  
  
Dim oFilletEdge As Edge  
Dim oBaseFace As Face  
Set oBaseFace = oBaseExtrude.EndFaces.Item(1)  
  
' ベース側とボス側間のエッジを見つける  
Dim oCheckEdge As Edge  
Set oCheckEdge = oSideFace.Edges.Item(1)  
  
If oCheckEdge.Faces(1) Is oBaseFace Or oCheckEdge.Faces(2) Is oBaseFace Then  
    Set oFilletEdge = oCheckEdge  
Else  
    Set oFilletEdge = oSideFace.Edges.Item(2)  
End If
```

# 演習8

## B-Rep によるアクセスとフィレットの作成

7. ベース側とボス側間のフィレットフィーチャーの為の、エッジコレクションを用意し、ベース側とボス側間のフィレットフィーチャーを新規作成した後でトランザクションを終了する。

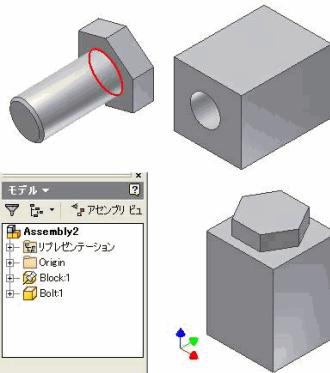
```
' ベース側とボス側間のフィレットフィーチャーの為の、エッジコレクションを用意.  
Dim oFilletEdges As EdgeCollection  
Set oFilletEdges = ThisApplication.TransientObjects.CreateEdgeCollection  
  
oFilletEdges.Add oFilletEdge  
  
' ベース側とボス側間のフィレットフィーチャーを新規作成。  
Dim oFillet As FilletFeature  
Set oFillet = _  
oDoc.ComponentDefinition.Features.FilletFeatures.AddSimple(oFilletEdges, 0.5)  
  
'トランザクションの終了  
oTrans.End  
  
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

# 演習9

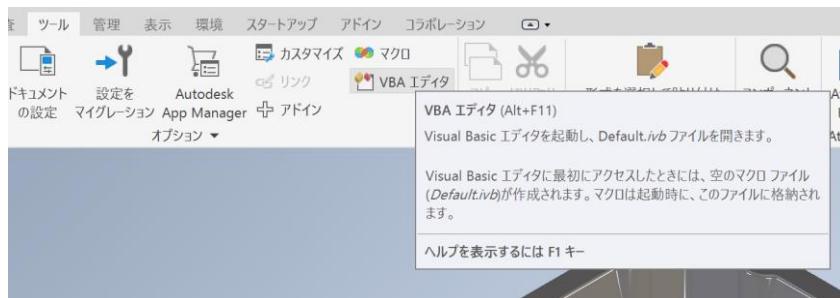
## アトリビュートを利用して拘束を持つアセンブリファイルの作成

アセンブリ内にボルトパートとブロックパートを挿入し、ブロック側はB-Rep APIを使って穴のエッジを見つけ、ボルト側はアトリビュートでオブジェクトを探し出し、両方のエッジを使ってパート間で挿入拘束を作成する。



事前準備: 新規にアセンブリファイルを開いておいてください。  
Inventor API Training\Drawings配下のBlock.iptとBolt.iptを使います。

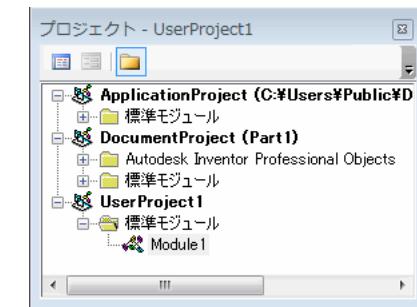
### 1. VBA エディタを起動します



### 2. 新しいプロジェクトを作成します



### 3. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習9

## アтриビュートを利用して拘束を持つアセンブリファイルの作成

4. Bolt.iptを開いて、ボルトの根元にあたるエッジに、AddAttributeマクロを用いて "BoltEdge"というアтриビュートを付加し、Bolt.iptファイルを保存します。

```
Public Sub AddAttribute()
    Dim oEdge As Edge
    On Error Resume Next

    Set oEdge = ThisApplication.ActiveDocument.SelectSet.Item(1)
    If Err Then
        MsgBox "エッジの選択が必要です"
        Exit Sub
    End If

    On Error GoTo 0
    ' 後で使用するために
    ' "BoltEdge"というアтриビュートを付加します。
    Call oEdge.AttributeSets.Add("BoltEdge")

End Sub
```

5. アセンブリファイル内に、“C:\Inventor API Training\Drawings”配下のBlock.iptとBolt.iptを使ってオカレンスを配置する。

```
Public Sub CreateAssemblyWithAttribute()
    Dim oAsmDoc As AssemblyDocument
    Set oAsmDoc = ThisApplication.ActiveDocument

    Dim oAsmDef As AssemblyComponentDefinition
    Set oAsmDef = oAsmDoc.ComponentDefinition

    Dim oTG As TransientGeometry
    Set oTG = ThisApplication.TransientGeometry

    'Block.iptとBolt.iptファイルを使ってオカレンスの配置
    Dim oBlockOcc As ComponentOccurrence
    Set oBlockOcc = oAsmDefOccurrences.Add("C:\Inventor API
Training\Drawings\Block.ipt", oTG.CreateMatrix)

    Dim oBoltOcc As ComponentOccurrence
    Set oBoltOcc = oAsmDefOccurrences.Add("C:\Inventor API
Training\Drawings\Bolt.ipt", oTG.CreateMatrix)
```

# 演習9

## アトリビュートを利用して拘束を持つアセンブリファイルの作成

6. Blockオカレンス側のシリンダーフェースをB-Repを使って辿り、  
ブロックの穴のエッジを見つけ 2つ目のループの内部ループを見  
つける。

'パートのシリンダーフェースを見てブロックの穴のエッジを見つけ、  
'2つ目のループの内部ループを見つける

```
Dim oFace As Face
Dim oBlockEdge As EdgeProxy

For Each oFace In oBlockOcc.SurfaceBodies.Item(1).Faces
    If oFace.SurfaceType = kCylinderSurface Then
        If Not oFace.EdgeLoops.Item(1).IsOuterEdgeLoop Then
            Set oBlockEdge = oFace.EdgeLoops.Item(1).Edges.Item(1)
        Else
            Set oBlockEdge = oFace.EdgeLoops.Item(2).Edges.Item(1)
        End If
    End If
```

```
    Exit For
End If
Next
```

7. Bolt側のオカレンスから、アトリビュートへの問い合わせでエッジオブジェクトを得る。エッジのプロキシはその時に構成される必要がある。

'ボルト側のドキュメントからボルトのエッジを得て、アトリビュートへの問い合わせでエッジを得る。  
'エッジのプロキシはその時に構成される必要がある。

```
Dim oBoltDoc As PartDocument
Set oBoltDoc = oBoltOcc.Definition.Document
```

```
' "BoltEdge"を持つアトリビュートのオブジェクトを得る
Dim oObjs As ObjectCollection
Set oObjs = oBoltDoc.AttributeManager.FindObjects("BoltEdge")

Dim oBoltEdge As EdgeProxy
Call oBoltOcc.CreateGeometryProxy(oObjs.Item(1), oBoltEdge)
```

8. 得られた2つのオブジェクトを使って、挿入拘束を作成する。

```
'拘束の作成。
Call oAsmDef.Constraints.AddInsertConstraint(oBlockEdge, oBoltEdge, True, 0)
```

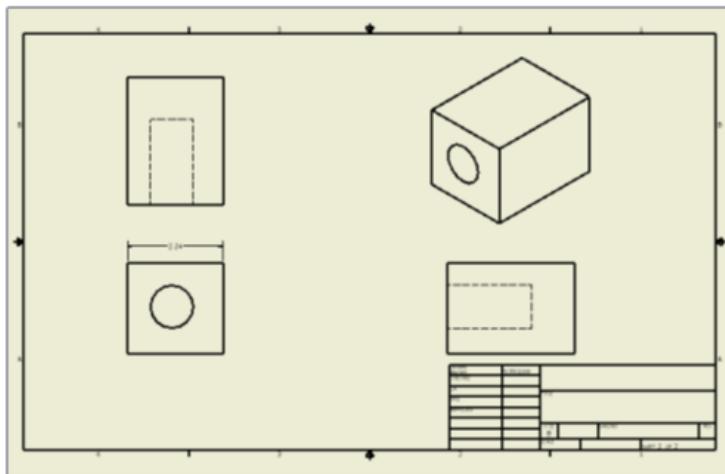
```
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。

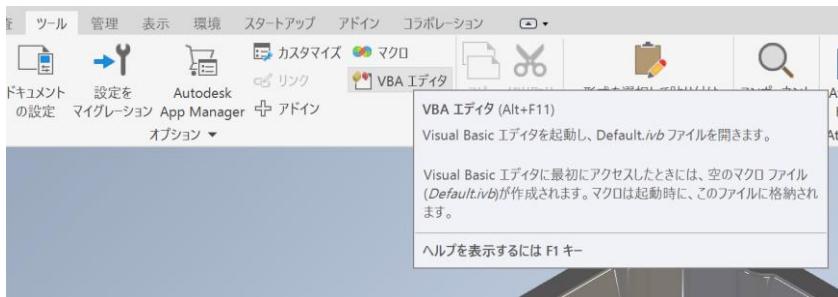
# 演習10

## シートとビューを持つ図面の作成

新規にDrawingドキュメントを作成し、既定の図面枠とJIS名のタイトルブロックを持つBサイズのシートを作成し、"Block.ipt"の正面、右、上面、等角ビューを作成する。正面ビュー内を調べて、トップの水平曲線を見つけ寸法を作成する。



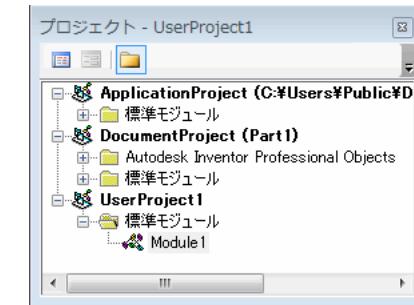
### 1. VBA エディタを起動します



### 2. 新しいプロジェクトを作成します



### 3. プロジェクト ウィンドウで、Module1 をダブルクリックします。編集ウィンドウが開きます。ここでプログラミングを開始します



# 演習10

## シートとビューを持つ図面の作成

4. 新規にDrawingドキュメントを作成し、既定の図面枠とJIS名のタイトルブロックを持つBサイズのシートを作成する。

```
Public Sub CreateDrawing()
    ' 新規Drawingドキュメントの作成。
    Dim oDoc As DrawingDocument
    Set oDoc = ThisApplication.Documents.Add(kDrawingDocumentObject, _
        ThisApplication.FileManager.GetTemplateFile(kDrawingDocumentObject))

    ' Bサイズシートの作成。
    Dim oSheet As Sheet
    Set oSheet = oDoc.Sheets.Add(kBDrawingSheetSize)

    ' 規定のボーダーの追加。
    oSheet.AddDefaultBorder

    ' JISタイトルブロックの追加。
    Call oSheet.AddTitleBlock(oDoc.TitleBlockDefinitions.Item("JIS"))
```

5. " C:\Inventor API Training\Drawings\Block.ipt "を目に見えないようにOpenし、正面、右、上面、等角ビューを作成する。

```
' 見えないように"Block.ipt"をオープン。
Dim oBlockPart As PartDocument
Set oBlockPart = ThisApplication.Documents.Open(_
    "C:\Inventor API Training\Drawings\Block.ipt", False)

Dim oTG As TransientGeometry
Set oTG = ThisApplication.TransientGeometry

' ベースビューの作成。
Dim oBaseView As DrawingView
Set oBaseView = oSheet.DrawingViews.AddBaseView(oBlockPart, _
    oTG.CreatePoint2d(10, 10), 1, kFrontViewOrientation, _
    kHiddenLineDrawingViewStyle)

' 他のビューを作成。
Dim oRightView As DrawingView
Set oRightView = oSheet.DrawingViews.AddProjectedView(oBaseView, _
    oTG.CreatePoint2d(30, 10), _
    kFromBaseDrawingViewStyle)

Dim oTopView As DrawingView
Set oTopView = oSheet.DrawingViews.AddProjectedView(oBaseView, _
    oTG.CreatePoint2d(10, 20), _
    kFromBaseDrawingViewStyle)
Dim oIsometricView As DrawingView
Set oIsometricView = oSheet.DrawingViews.AddProjectedView(oBaseView, _
    oTG.CreatePoint2d(30, 20), _
    kHiddenLineRemovedDrawingViewStyle)
```

# 演習10

## シートとビューを持つ図面の作成

### 6. 同じ点か否かの判定をするWithinTol関数を作成する

```
Private Function WithinTol(ByVal Value1 As Double, ByVal Value2 As Double) As Boolean
    If Abs(Value1 - Value2) < 0.0001 Then
        WithinTol = True
    Else
        WithinTol = False
    End If
End Function
```

### 7. 作図ビューを調べて、トップの水平曲線を見つける。(寸法の為にパーツ内のエッジを探す)。

```
' 寸法のためにパーツ内のエッジを探す。どんなメソッドも使用可能
' (attributes, B-Rep query, selection, etc.).
' 以下は、作図ビューを調べて、トップの水平曲線を見つける。
Dim oCurve As DrawingCurve
Dim oDimCurve As DrawingCurve

For Each oCurve In oBaseView.DrawingCurves
    ' 円はスキップ。
    If Not oCurve.StartPoint Is Nothing And Not oCurve.EndPoint Is Nothing Then
        If WithinTol(oCurve.StartPoint.Y, oCurve.EndPoint.Y) Then
            If oDimCurve Is Nothing Then
                ' 最初の水平曲線が見つけ、保存。
                Set oDimCurve = oCurve
            Else
                ' カーブがより高い(Y値が大きい)場合そちらを有効として保存するためのチェック。
                Dim oCurve1 As Double
                Dim oCurve2 As Double

                oCurve1 = oCurve.MidPoint.Y
                oCurve2 = oDimCurve.MidPoint.Y
                If oCurve1 > oCurve2 Then
                    Set oDimCurve = oCurve
                End If
            End If
        End If
    End If
End If
End If
End If
Next
```

# 演習10

## シートとビューを持つ図面の作成

8. 曲線のためのGeometryIntentポイントを作成し、直線寸法を作成する。

```
' カーブのためのGeometryIntentポイントの作成。  
' シングルラインとして直線寸法を置く。  
' (寸法が取り付けられるラインに対し、位置を定義する必要がある)  
Dim oGeomIntent As GeometryIntent  
Set oGeomIntent = oSheet.CreateGeometryIntent(oDimCurve)  
  
' 寸法の作成。  
Dim oLinearDim As LinearGeneralDimension  
Set oLinearDim = oSheet.DrawingDimensions.GeneralDimensions.AddLinear_  
    (oTG.CreatePoint2d(oDimCurve.MidPoint.X, oDimCurve.MidPoint.Y + 1),  
     oGeomIntent, , kAlignedDimensionType)  
  
End Sub
```

演習を完成させるのが難しい場合は、「演習の解答」にある完成済みのコードを参照してください。



AUTODESK®

Make anything™