

## Lab 6 – 拡張ストレージ

Created by A. Nagy, May 2011

Updated by DevTech AEC WG

Last modified: 6/9/2017

### <VB.NET>VB.NET バージョン</VB.NET>

**目的:**この実習では、カスタム・データを Revit の要素に加えるために拡張ストレージを使用する方法を学習します。さらに、コマンドの実装過程で、選択フィルタとトランザクションを習得していきます。学習する項目は次のとおりです。

- ストレージ データとなる「スキーマ」を定義して、Revit 要素にそのインスタンスを付加する
- 選択する要素を壁に制限するために、ISelectionFilter を使用する
- 手動トランザクション モードを使用する

**タスク:** 2 つのフィールド(SocketLocation と SocketNumber)で WallSocketLocation スキーマを作成して、そのエンティティ(実体、ないしはインスタンス)を壁に付加します。

1. ユーザに壁を選択するよう促す
2. 2 つのフィールドを持つスキーマを作成する
3. スキーマのエンティティ(実体)を作成し、そのフィールド値を設定する
4. SetEntity()を使用して、壁にエンティティ(実体)を割り当てる

この実習の実装と確認の手順は、下記のとおりです:

1. 手動トランザクションモードで新しい外部コマンドを定義する
2. 選択フィルタを作成する
3. スキーマのエンティティ(実体)を作成して壁に付加する
4. サマリ

## 1. 手動トランザクション モードで新しい外部コマンドを定義する

現在のプロジェクトに新しい外部コマンドを追加します。

1.1 新しいファイルを追加して、プロジェクトに新しい外部コマンドを定義します。ファイル名とクラス名は、下記のようにしてください:

- ファイル名: **6\_ExtensibleStorage.vb**
- コマンド クラス名: **ExtensibleStorage**

要求される名前空間:

いままで使用した名前空間に加えて、次の名前空間を加えてください:

- Autodesk.Revit.DB.ExtensibleStorage

下記は、新しいコマンドの開始点です。

```
<VB.NET>
'   Model Creation - learn how to create elements

<Transaction(TransactionMode.Manual)> _
Friend Class ExtensibleStorage
    Implements IExternalCommand

    Public Function Execute( _
        ByVal commandData As ExternalCommandData, _
        ByRef message As String, _
        ByVal elements As ElementSet) _
        As Result _
        Implements IExternalCommand.Execute

        Dim uiDoc As UIDocument = commandData.Application.ActiveUIDocument
        Dim doc As Document = uiDoc.Document

        ' Create transaction for working with schema

        Dim trans As New Transaction(doc, "Extensible Storage")
        trans.Start()

        ' ...

        trans.Commit()

        Return Result.Succeeded
    End Function
```

```
End Class
```

```
</VB.NET>
```

## 2. 選択フィルタを作成する

ここでは、PickObject() と ISelectionFilter を使用しますが、それらの細かい説明は UI 実習の中で行います。まずは、ISelectionFilter インターフェースの実装クラスを作成する必要があります。この実装クラスは、2つの関数 AllowElement () と AllowReference() を持ちます。ExtensibleStorage クラス内で、これらを作成します:

```
<VB.NET>
```

```
Private Class WallSelectionFilter
    Implements ISelectionFilter

    Public Function AllowElement(ByVal e As Element) As Boolean _
        Implements ISelectionFilter.AllowElement

        Return TypeOf e Is Wall

    End Function

    Public Function AllowReference(ByVal r As Reference, ByVal p As XYZ) _
        As Boolean Implements ISelectionFilter.AllowReference

        Return True

    End Function
End Class
```

```
End Class
```

```
</VB.NET>
```

ユーザ インターフェイス上で、ユーザに要素の選択を促す際に、選択対象を壁だけに制限するために、上記のクラスのインスタンスを使用します。

ユーザに、データを追加したい壁を選択するように促して、返された参照から壁を取得します。:

```
<VB.NET>
```

```
' Pick a wall

Dim r As Reference = uiDoc.Selection.PickObject( _
    ObjectType.Element, New WallSelectionFilter)

Dim wall As Wall = TryCast(doc.GetElement(r), Wall)
```

```
</VB.NET>
```

### 3. スキーマを作成して、そのエンティティ(実体)を壁に付加する

スキーマを作成するためには、後からスキーマを識別するためのGUIDが必要となります。このGUID を使用して、SchemaBuilder オブジェクトを作成します。ExtensibleStorage クラスでこのGUIDを宣言しましょう:

```
<VB.NET>
Private _guid As Guid = New Guid("87aaad89-6f1b-45e1-9397-2985e1560a02")
</VB.NET>
```

注意: Visual Studio のツール([GUIDを生成])を使って、同様に新しいGUIDを生成して利用することもできます。

新しいSchemaBuilderインスタンスを作成することでスキーマの構築を始めていきます。今回は、読み込みと書き込みのアクセス レベルをパブリックに設定します。それをベンダーまたはアプリケーションに設定する場合には、SetVendorId()を使用して、スキーマ用にベンダーIdを指定する必要があります。ベンダーIdは、次の URL で作成することができるRegistered Developer Symbol (RDS)です:

<http://www.autodesk.com/symbreg>

ベンダーid は大文字・小文字を区別しないので、文字列はスキーマに格納される前に大文字に変換されるでしょう。

```
<VB.NET>
' Create a scheme builder

Dim builder As New SchemaBuilder(_guid)

' Set read and write access levels

builder.SetReadAccessLevel(AccessLevel.Public)
builder.SetWriteAccessLevel(AccessLevel.Public)

' Note: if this was set as vendor or application access,
' we would have been additionally required to use SetVendorId

' Set name to this schema builder

builder.SetSchemaName("WallSocketLocation")
builder.SetDocumentation("Data store for socket info in a wall")
</VB.NET>
```

2つのフィールドをスキーマに追加します。1つは位置プロパティを含む XYZ タイプになります。もう1つは、ソケットのid番号を含む文字列タイプになります。一旦これらが追加されたら、スキーマ オブジェクトを作成するためにFinish()を呼び出すことができます。

```
<VB.NET>
' Create field1

Dim fieldBuilder1 As FieldBuilder = _
    builder.AddSimpleField("SocketLocation", GetType(XYZ)). _
    SetUnitType(UnitType.UT_Length)
```

```

' Set unit type

fieldBuilder1.SetUnitType(UnitType.UT_Length)

' Add documentation (optional)

' Create field2

Dim fieldBuilder2 As FieldBuilder = _
    builder.AddSimpleField("SocketNumber", GetType(String))

'fieldBuilder2.SetUnitType(UnitType.UT_Custom);

' Register the schema object

Dim schema As Schema = builder.Finish
</VB.NET>

```

ここで、スキーマに基づいた2つのエンティティ(実体)を作成して、選択された壁にそれらを割り当てます。

```

<VB.NET>
' Create an entity (object) for this schema (class)

Dim ent As New Entity(schema)

Dim socketLocation As Field = schema.GetField("SocketLocation")
ent.Set(Of XYZ)(socketLocation, New XYZ(2, 0, 0), _
    DisplayUnitType.DUT_METERS)

Dim socketNumber As Field = schema.GetField("SocketNumber")
ent.Set(Of String)(socketNumber, "200")

wall.SetEntity(ent)

' Now create another entity (object) for this schema (class)
' (This simply replaces the ent1 above. Just for testing.
' You may comment out for now.)

Dim ent2 As New Entity(schema)
Dim socketNumber1 As Field = schema.GetField("SocketNumber")
ent2.Set(Of String)(socketNumber1, "400")
wall.SetEntity(ent2)
</VB.NET>

```

さらに、期待した設定が出来たのか参照するために、利用可能なすべてのスキーマと、スキーマ内で利用可能なフィールドをリストします。

```

<VB.NET>
' List all schemas in the document

Dim s As String = String.Empty
Dim schemas As IList(Of Schema) = schema.ListSchemas

```

```

Dim sch As Schema
For Each sch In schemas
    s += vbCrLf + "Schema name: " + sch.SchemaName
Next
TaskDialog.Show("Schema details", s)

' List all Fields for our schema

s = String.Empty
Dim fields As IList(Of Field) = schema.Lookup(_guid).ListFields
Dim fld As Field
For Each fld In fields
    s += vbCrLf + "Field name: " + fld.FieldName
Next
TaskDialog.Show("Field details", s)
</VB.NET>

```

選択した壁に割り当てたエンティティ(実体)が、正しく割り当てられたかチェックします。

```

<VB.NET>
' Extract the value for the field we created

Dim wallSchemaEnt As Entity = wall.GetEntity(schema.Lookup(_guid))

Dim wallSocketPos As XYZ = wallSchemaEnt.Get(Of XYZ) ( _
    schema.Lookup(_guid).GetField("SocketLocation"), _
    DisplayUnitType.DUT_METERS)

s = "SocketLocation: " + PointToString(wallSocketPos)

Dim wallSocketNumber As String = wallSchemaEnt.Get(Of String) ( _
    schema.Lookup(_guid).GetField("SocketNumber"))

s += vbCrLf + "SocketNumber: " + wallSocketNumber

TaskDialog.Show("Field Values", s)
</VB.NET>

```

下記の PointToString() は、点座標を表示用の文字列に変換する簡単なヘルパー関数です。Lab2 内で記述したものを再利用することも可能です。例えば:下記のようになります:

```

<VB.NET>

'' Helper Function: returns XYZ in a string form.
''

Public Shared Function PointToString(ByVal pt As XYZ) As String

    If pt Is Nothing Then
        Return ""
    End If

    Return "(" + pt.X.ToString("F2") + ", " + pt.Y.ToString("F2") + _

```

```
    ", " + pt.Z.ToString("F2") + ")"  
End Function  
</VB.NET>
```

## 4. サマリ

この実習では、API の拡張ストレージを使用して、カスタム データのスキーマを定義し、そのエンティティ(実体)を Revit の要素へ付加する方法を学習しました。学習した内容は次のとおりです。

- Revit 要素の拡張ストレージのエンティティを作成してアクセスする

Autodesk Developer Network