

Lab 6 – 拡張ストレージ

Created by A. Nagy, May 2011
Updated by Ryuji Ogasawara
Last modified: 8/8/2025

<C#>C#バージョン</C#>

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
- ユーザが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.cs**
- コマンド クラス名: **ExtensibleStorage**

要求される名前空間:

これ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:

```
<C#>
// Model Creation - learn how to create elements

[Transaction(TransactionMode.Manual)]
public class ExtensibleStorage : IExternalCommand
{
    public Result Execute(
        ExternalCommandData commandData,
        ref string message,
        ElementSet elements)
    {
        // Get the access to the top most objects.
        UIDocument uiDoc = commandData.Application.ActiveUIDocument;
        Document doc = uiDoc.Document;

        Transaction trans = new Transaction(doc, "Extensible Storage");
        trans.Start();

        // ...

        trans.Commit();

        return Result.Succeeded;
    }
}
```

```
}  
</C#>
```

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、これらの詳細な説明は 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:

```
<C#>  
class WallSelectionFilter : ISelectionFilter  
{  
    public bool AllowElement( Element e )  
    {  
        return e is Wall;  
    }  
  
    public bool AllowReference( Reference r, XYZ p )  
    {  
        return true;  
    }  
}  
</C#>
```

ユーザ インターフェース上で、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。:

```
<C#>  
    // Pick a wall  
  
    Reference r = uiDoc.Selection.PickObject(  
        ObjectType.Element, new WallSelectionFilter());  
  
    Wall wall = doc.GetElement(r) as Wall;  
</C#>
```

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:

```
<C#>
Guid _guid = new Guid("87aaad89-6f1b-45e1-9397-2985e1560a02");
</C#>
```

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。メニューに表示されない場合は、外部ツールから、下記のコマンドを追加します。

C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\Common7\Tools\guidgen.exe

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。

```
<C#>
    // Create a schema builder

    SchemaBuilder builder = 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");
</C#>
```

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。

```
<C#>
    // Create field1

    FieldBuilder fieldBuilder1 =
        builder.AddSimpleField("SocketLocation", typeof(XYZ));

    // Set unit type

    fieldBuilder1.SetUnitType(UnitType.UT_Length);

    // Add documentation (optional)

    // Create field2
```

```

FieldBuilder fieldBuilder2 =
    builder.AddSimpleField("SocketNumber", typeof(string));

//fieldBuilder2.SetUnitType(UnitType.UT_Custom);

// Register the schema object

Schema schema = builder.Finish();
</C#>

```

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。

```

<C#>
    // Create an entity (object) for this schema (class)

    Entity ent = new Entity(schema);

    Field socketLocation = schema.GetField("SocketLocation");
    ent.Set<XYZ>(socketLocation, new XYZ(2, 0, 0),
        DisplayUnitType.DUT_METERS);

    Field socketNumber = schema.GetField("SocketNumber");
    ent.Set<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.)

    Entity ent2 = new Entity(schema);
    Field socketNumber1 = schema.GetField("SocketNumber");
    ent2.Set<String>(socketNumber1, "400");
    wall.SetEntity(ent2);
</C#>

```

さらに、期待した設定が出来たか確認するために、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。

```

<C#>
    // List all schemas in the document

    string s = string.Empty;
    IList<Schema> schemas = Schema.ListSchemas();
    foreach( Schema sch in schemas )
    {
        s += "\r\nSchema Name: " + sch.SchemaName;
    }
    TaskDialog.Show( "Schema details", s );

    // List all Fields for our schema

```

```

s = string.Empty;
Schema ourSchema = Schema.Lookup( _guid );
IEnumerable<Field> fields = ourSchema.ListFields();
foreach( Field fld in fields )
{
    s += "\r\nField Name: " + fld.FieldName;
}
TaskDialog.Show( "Field details", s );
</C#>

```

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。

```

<C#>
// Extract the value for the field we created

Entity wallSchemaEnt = wall.GetEntity( Schema.Lookup( _guid ) );

XYZ wallSocketPos = wallSchemaEnt.Get<XYZ>(
    Schema.Lookup( _guid ).GetField( "SocketLocation" ),
    DisplayUnitType.DUT_METERS );

s = "SocketLocation: " + PointToString( wallSocketPos );

string wallSocketNumber = wallSchemaEnt.Get<String>(
    Schema.Lookup( _guid ).GetField( "SocketNumber" ) );

s += "\r\nSocketNumber: " + wallSocketNumber;

TaskDialog.Show( "Field values", s );
</C#>

```

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:下記ようになります:

```

<C#>
// Helper Function: returns XYZ in a string form.
//
public static string PointToString(XYZ pt)
{
    if (pt == null)
    {
        return "";
    }

    return string.Format("{0},{1},{2}",
        pt.X.ToString("F2"), pt.Y.ToString("F2"), pt.Z.ToString("F2"));
}
</C#>

```

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

拡張ストレージの詳細な解説は、Revit API 開発者用ガイドの「[拡張ストレージ](#)」項目をご参照ください。

Autodesk Developer Network