

Lab 2 – 選択

May, 2011 by A. Nagy
Updated by DevTech AEC WG
Last modified: 8/4/2015

<C#>C#バージョン</C#>

目的:この実習では、様々な選択機能を使用する方法を学習します。学習する項目は次のとおりです。

- オブジェクト/オブジェクト群/点(座標)を選択する
- 要素ジオメトリを選択する
- 選択を制限する

この実習の実装と確認の手順は、下記のとおりです:

1. 新しい外部コマンドを作成する
2. 現在の選択内容を示す
3. オブジェクトを選択する
4. オブジェクト群を選択する
5. 点を選択する
6. 面/エッジを選択する
7. 選択フィルタ
8. 対話的に家を作成する
9. サマリ

1. 新しい外部コマンドを作成する

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: **2_Selection.cs**
- コマンド クラス名: **UISelection**

(繰り返しになりますが、ここで希望する名前を使用しても構いません。ただし、その場合、プロジェクト名など、このドキュメント内では記述されている名称は、自分でつけた名称で代替して参照してください)

要求される名前空間:

この実習に必要とされる名前空間は次のとおりです:

- System.Collections
- System.Collections.Generic
- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.ApplicationServices
- Autodesk.Revit.Attributes
- Autodesk.Revit.UI.Selection(選択処理用)

注意: (VB.NET のみ):VB.NET でコードを記述している場合、プロジェクト・レベルで名前空間をインポートします(つまり、プロジェクトプロパティでは、各ファイル内で明示的にインポートする必要はありません)。

クラス内で UIApplication とアクティブな UIDocument を参照する変数を宣言します。

```
<C#>
[Transaction(TransactionMode.ReadOnly)]
public class UISelection : IExternalCommand
{
    UIApplication _uiApp;
    UIDocument _uiDoc;

    public Result Execute(
        ExternalCommandData commandData,
```

```

        ref string message,
        ElementSet elements )
    {
        _uiApp = commandData.Application;
        _uiDoc = _uiApp.ActiveUIDocument;

        // ...
        return Result.Succeeded;
    }
</C#>

```

2. 現在の選択内容を示す

現在選択されているオブジェクトの内容に関する情報を示すために、いくつかのヘルパー関数を作成しましょう。これらのヘルパー関数は、他の関数から呼び出して使用します。

```

<C#>
public void ShowElementList(IEnumerable elems, string header)
{
    string s = "\n\n - Class - Category - Name (or Family: Type Name) - Id
- " + "\r\n";

    int count = 0;
    foreach (Element e in elems)
    {
        count++;
        s += ElementToString( e );
    }

    s = header + "(" + count + ")" + s;

    TaskDialog.Show( "Revit UI Lab", s );
}

public string ElementToString(Element e)
{
    if( e == null )
    {
        return "none";
    }

    string name = "";

    if( e is ElementType )
    {
        Parameter param = e.get_Parameter(
            BuiltInParameter.SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM );
        if( param != null )
        {

```

```

        name = param.AsString();
    }
}
else
{
    name = e.Name;
}

return e.GetType().Name + "; " + e.Category.Name + "; " + name + "; "
    + e.Id.IntegerValue.ToString() + "\r\n";
}

public static string PointToString(XYZ pt)
{
    if( pt == null )
    {
        return "";
    }

    return "(" + pt.X.ToString( "F2" ) + ", " + pt.Y.ToString( "F2" ) + ",
"
        + pt.Z.ToString( "F2" ) + ")";
}
</C#>

```

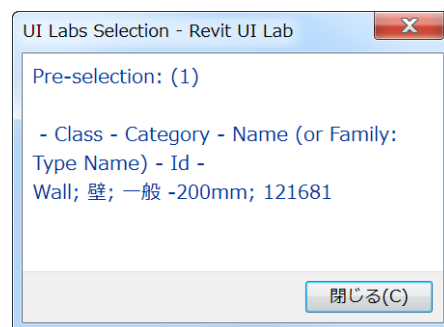
UIDocument.Selection.GetElementIds() から、現在選択された要素 ID のリストを取得できます。(注意: Revit 2015 から UIDocument.Selection.Elements は非推奨です)

```

<C#>
ICollection<ElementId> selectedElementIds = _uiDoc.Selection.GetElementIds();
ShowElementList(selectedElementIds, "Pre-selection: ");
</C#>

```

3. オブジェクトを選択する



様々な選択関数は、すべて UIDocument.Selection 下で見つけることができます。まず、ユーザに単一のオブジェクトの選択を促す PickObject() を最初に見ていきます。この関数には、多くのオーバーロードが存在します。選択を許可するオブジェクトのタイプを指定することも出来ます。今回は、選択対象を要素として、ユーザにプロンプトを提供します。さらに、選択フィルタ クラスを指定することも出来ます(後述)。

```
<C#>
public void PickMethod_PickObject()
{
    Reference r = _uiDoc.Selection.PickObject(
        ObjectType.Element, "Select one element" );

    Element e = _uiDoc.Document.GetElement( r );

    ShowBasicElementInfo( e );// Defined in DBElement Lab in the Intro Labs
}
</C#>
```

4. オブジェクト群を選択する

複数のオブジェクトを選択するようにユーザに促したい場合には、選択したオブジェクトの参照のリストを返す PickObjects() を呼び出してください。先に作成した、選択要素に関する情報を示すユーティリティ関数を使用するために、参照のリストを要素のリストに変換する必要があります。

```
<C#>
public void PickMethod_PickObjects()
{
    IList<Reference> refs = _uiDoc.Selection.PickObjects(
        ObjectType.Element, "Select multiple elements" );

    // Put it in a List form.

    IList<Element> elems = new List<Element>();
    foreach( Reference r in refs )
    {
        elems.Add( _uiDoc.Document.GetElement( r ) );
    }

    ShowElementList( elems, "Pick Objects: " );
}
</C#>
```

さらに、矩形内にある要素を選択するようにユーザに促すことも出来ます。

```
<C#>
public void PickMethod_PickElementByRectangle()
{
    // Note: PickElementByRectangle returns the list of element.
    // not reference.
    IList<Element> elems = _uiDoc.Selection.PickElementsByRectangle(
        "Select by rectangle" );
}
```

```

        // Show it.

        ShowElementList( elems, "Pick By Rectangle: " );
    }
</C#>

```

5. 点を選択する

空間上の点(座標)を選択したい場合には、PickPoint() を使用することが出来ます。

```

<C#>
public void PickMethod_PickPoint()
{
    XYZ pt = _uiDoc.Selection.PickPoint( "Pick a point" );

    // Show it.

    string msg = "Pick Point: ";
    msg += PointToString( pt );

    TaskDialog.Show( "PickPoint", msg );
}
</C#>

```

要素上の点(座標)を選択したければ、PickObject() に ObjectType.PointOnElement を渡します。

```

<C#>
public void PickPointOnElement()
{
    Reference r = _uiDoc.Selection.PickObject(
        ObjectType.PointOnElement,
        "Select a point on element" );

    Element e = _uiDoc.Document.GetElement( r );
    XYZ pt = r.GlobalPoint;

    string msg = "";
    if( pt != null )
    {
        msg = "You picked the point " + PointToString( pt )
            + " on an element " + e.Id.ToString() + "\r\n";
    }
    else
    {
        msg = "no Point picked \n";
    }

    TaskDialog.Show( "PickPointOnElement", msg );
}
</C#>

```

6. 面/エッジを選択する

PickObject() に、ObjectType.Face または ObjectType.Edge をそれぞれ渡すことで、要素の面やエッジを選択することができます。これによって、要素の参照が返されます。選択した要素の面やエッジを得るためには、GetGeometryObjectFromReference() を使用します。

```
<C#>
public void PickFace()
{
    Reference r = _uiDoc.Selection.PickObject( ObjectType.Face,
        "Select a face" );
    Element e = _uiDoc.Document.GetElement( r );

    Face oFace = e.GetGeometryObjectFromReference( r ) as Face;

    string msg = "";
    if( oFace != null )
    {
        msg = "You picked the face of element " + e.Id.ToString() + "\r\n";
    }
    else
    {
        msg = "no Face picked \n";
    }

    TaskDialog.Show( "PickFace", msg );
}
</C#>
```

エッジを選択するために、同様の関数を実装してみてください。

7. 選択フィルタ

PickObject()/PickObjects() を使用してオブジェクトを選択した場合、希望する方法で選択をフィルタリングするために、ISelectionFilter を実装するクラスのインスタンスを指定することもできます。例えば、ユーザが壁だけを選択するように、選択操作をフィルタリングできます。

最初に、ISelectionFilter インターフェースを実装するクラスを作成します。このインターフェースは、実装可能な2つのメソッド、AllowReference () と AllowElement() があります。ObjectType.Element を PickObject() に指定すれば、AllowElement() だけが呼び出されるようになります。もし、ObjectType.Face/Edge/PointOnElement を指定すれば、AllowReference() が呼び出されます。

```
<C#>
class SelectionFilterWall : ISelectionFilter
{
    public bool AllowElement( Element e )
    {
        return e is Wall;
    }
}
```

```

public bool AllowReference( Reference reference, XYZ position )
{
    return true;
}
}
</C#>

```

このクラスを使用して、選択をフィルタリングしてみましょう:

```

<C#>
public void PickWall()
{
    SelectionFilterWall selFilterWall = new SelectionFilterWall();
    Reference r = _uiDoc.Selection.PickObject( ObjectType.Element,
        selFilterWall, "Select a wall" );

    // Show it
    Element e = _uiDoc.Document.GetElement( r );

    ShowBasicElementInfo( e );// Defined in DBElelemt Lab in the Intro Labs
}
</C#>

```

面オブジェクトを選択する場合には、PlanarFace で選択をフィルタリングしてみてください。

8. 対話的に家を作成する

入門実習には、家を作成する関数をエクスポートする部分が含まれています。入門実習を開いてビルドして、現在のプロジェクトに、作成されたアドインの実行ファイル dll への参照を追加してください。参照される入門実習 dll の ModelCreationExport クラスの下には、対話的に家を作る手助けとなる様々な関数があります。それらの関数は、次のとおりです:

- CreateWalls()
- AddDoor()
- AddWindow()
- AddRoof()

下記の処理を実行するプログラムを作成してください:

- 新しい外部コマンドを 2_Selection.cs ファイルに加える
- コマンド プロンプト内で家の 2 つの対角点を選択して、それらの点に基づく 4 つの壁を作成する
- 壁のうちの 1 つを選択するようにユーザに促して、そこにドアを追加し、窓を他の 3 つの壁の各々に加える

- 家の屋根を作成する

9. サマリ

この実習では、様々な選択機能を使用する方法を学習しました。学習した項目は次のとおりです。

- オブジェクト/オブジェクト/ポイントを選択する
- 要素ジオメトリを選択
- 選択を制限する