

## Lab3 – 要素フィルタリング

Created by M. Harada, July 2010

Updated by Ryuji Ogasawara

Last modified: 5/30/2024

<C#>C#バージョン</C#>

**目的:**この実習では、Revit API のフィルタリング メカニズムを使用して、特定の要素を絞り込む方法を学習します。学習する項目は次のとおりです。

- ファミリタイプを取得する
- 特定のオブジェクトクラスのインスタンスを取得する
- 特定のファミリタイプを見つける
- 特定インスタンスを見つける

**タスク:**要素のフィルタリングの様々な手法とアプローチを理解しながらコマンドを記述していきます。フィルタリングを習熟するために、この実習を使用してください。

1. ファミリタイプをリストする(例、壁タイプ、床タイプやドアタイプ)
2. 特定のタイプのオブジェクト インスタンスをリストする(例、壁とドア)
3. 特定のファミリタイプを見つける(例、“標準壁: 一般 -200mm” 壁タイプ、“片側フラッシュ: 0915 x 2134mm” ドアタイプ)
4. 特定のインスタンスを見つける(例、“標準壁: 一般 -200mm” 壁タイプのインスタンス、片側フラッシュ: 0915 x 2134mm” ドアタイプのインスタンス、一定長より長い壁)

図 1 は、この実習で定義するコマンドで表示されるサンプル画像を示します:

※日本語版 Revit で「建設テンプレート」を指定して新規プロジェクトを作成後、各コマンドを実行します。



図 1.様々なフィルタリングの結果を示すダイアログ

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定義する
2. ファミリ タイプをリストする
3. 特定オブジェクトクラスのインスタンスをリストする
4. 特定のファミリ タイプを見つける
5. 特定のインスタンスを見つける
6. サマリ

## 1. 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: **3\_ElementFiltering.cs**
- コマンド クラス名: **ElementFiltering**

(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記述されている名称を、自分でつけた名称に置き換えて参照してください)

**要求される名前空間:**

この実習で必要となる名前空間は次のとおりです:

- System.Linq
- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.ApplicationServices
- Autodesk.Revit.Attributes

1.2 Lab2 の中で行ったように、メンバ変数を定義します。例えば、DB レベルのアプリケーションとドキュメントをそれぞれ保持する `m_rvtApp` と `m_rvtDoc` を定義します。下記はその例です:

```
<C#>
// Element Filtering - learn about Revit element
[Transaction(TransactionMode.Manual)]
public class ElementFiltering : IExternalCommand
{
    // member variables
    Application m_rvtApp;
    Document m_rvtDoc;

    public Result Execute(
        ExternalCommandData commandData,
        ref string message,
        ElementSet elements)
    {
        // Get the access to the top most objects.
        UIApplication rvtUIApp = commandData.Application;
```

```

        UIDocument rvtUIDoc = rvtUIApp.ActiveUIDocument;
        m_rvtApp = rvtUIApp.Application;
        m_rvtDoc = rvtUIDoc.Document;

        // ...

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

```

## 2. ファミリタイプをリストする

前の実習では、要素がコンポーネントファミリに基づくのか、システム・ファミリに基づくのかによって、クラス名とカテゴリを使用して、要素を識別するために異なるアプローチをとる必要があるということ学習しました。プロジェクトデータベースに格納されたファミリタイプのリストを取得する場合にも、同様のルールが当てはまります。

### 2.1 システム・ファミリタイプ

Revit の要素を識別する際、それらの要素が大きな袋に入れられていて、その袋がデータベースにあると考えることができます。その中の要素にアクセスするために、クエリ(照会)をおこなう必要があります。例えば、下記のコードは、ドキュメント内のすべての WallType クラスを収集します:

```

<C#>
var wallTypeCollector1 = new FilteredElementCollector(m_rvtDoc);
wallTypeCollector1.WherePasses(new ElementClassFilter(typeof(WallType)));
IList<Element> wallTypes1 = wallTypeCollector1.ToElements();
</C#>

```

FilteredElementCollector は、私たちが絞り込む要素を収集する「コンテナ」オブジェクトで、最初にそれを作成する必要があります。また、この場合、クラスフィルタはフィルタを通して WallTypes クラスの要素だけを収集します。最終行は、フィルタされた要素コレクタを要素のリストに変換します。リストに変換することで、それ以降の要素を操作しやすくなります。

このサンプルコードでは、壁タイプの一覧を取得するためにフィルタを使用する利点を見いだせないかもしれません。しかし、クエリの内容がより複雑になる場合には、この方法がより便利になってくるはずです。

Revit APIは、様々なフィルタリング方式を提供しています。下記は、上記のコードの最初の2行と同じ内容ですが、OfClass() を使用しています:

```

<C#>
FilteredElementCollector wallTypeCollector2 =
    new FilteredElementCollector(m_rvtDoc);
wallTypeCollector2.OfClass(typeof(WallType));
</C#>

```

さらに、ショートカットを使用して、これを単純化することができます:

```
<C#>
    FilteredElementCollector wallTypeCollector3 =
        new FilteredElementCollector(m_rvtDoc).OfClass(typeof(WallType));
</C#>
```

## 2.3 コンポーネント ファミリ タイプ

コンポーネント ファミリ用には、ドキュメント クラスに指定のプロパティはありません。したがって、常にフィルタリングを使用する必要があります。コンポーネント ファミリでは、要素クラスとカテゴリをチェックする必要がありました。次のコードは、ドア ファミリ タイプの一覧を得る例です。

```
<C#>
var doorTypeCollector = new FilteredElementCollector(m_rvtDoc);
doorTypeCollector.OfClass(typeof(FamilySymbol));
doorTypeCollector.OfCategory(BuiltInCategory.OST_Doors);
IList<Element> doorTypes = doorTypeCollector.ToElements();
</C#>
```

次のコードで使用方法を確認しましょう:

```
<C#>
public void ListFamilyTypes()
{
    // (1) Get a list of family types available in the current rvt project.
    //
    // For system family types, there is a designated
    // properties that allows us to directly access to the types.
    // e.g., _doc.WallTypes

    //WallTypeSet wallTypes = _doc.WallTypes; // 2013

    FilteredElementCollector wallTypes
        = new FilteredElementCollector(_doc) // 2014
        .OfClass(typeof(WallType));
    int n = wallTypes.Count();

    string s = string.Empty;
    //int n = 0;

    foreach (WallType wType in wallTypes)
    {
        s += "\r\n" + wType.Kind.ToString() + " : " + wType.Name;
        //++n;
    }
    TaskDialog.Show(n.ToString()
        + " Wall Types:",
```

```

        s);

    // (1.1) Same idea applies to other system family, such as Floors,
    Roofs.

    //FloorTypeSet floorTypes = _doc.FloorTypes;

    FilteredElementCollector floorTypes
    = new FilteredElementCollector(_doc) // 2014
        .OfClass(typeof(FloorType));

    s = string.Empty;

    foreach (FloorType fType in floorTypes)
    {
        // Family name is not in the property for
        // floor, so use BuiltInParameter here.

        Parameter param = fType.get_Parameter(
            BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM);

        if (param != null)
        {
            s += param.AsString();
        }
        s += " : " + fType.Name + "\r\n";
    }
    TaskDialog.Show(
        floorTypes.Count().ToString() + " floor types (by rvtDoc.FloorTypes):",
        s);

    // (1.2a) Another approach is to use a filter. here is an example with
    wall type.

    var wallTypeCollector1 = new FilteredElementCollector(_doc);
    wallTypeCollector1.WherePasses(new
    ElementClassFilter(typeof(WallType)));
    IList<Element> wallTypes1 = wallTypeCollector1.ToElements();

    // Using a helper function to display the result here. See code below.

    ShowElementList(wallTypes1, "Wall Types (by Filter): ");

    // (1.2b) The following are the same as two lines above.
    // These alternative forms are provided for convenience.
    // Using OfClass()
    //
    //FilteredElementCollector wallTypeCollector2 = new
    FilteredElementCollector(_doc);
    //wallTypeCollector2.OfClass(typeof(WallType));

    // (1.2c) The following are the same as above for convenience.
    // Using short cut this time.

```

```

//
//FilteredElementCollector wallTypeCollector3 = new
FilteredElementCollector(_doc).OfClass(typeof(WallType));

//
// (2) Listing for component family types.
//
// For component family, it is slightly different.
// There is no designate property in the document class.
// You always need to use a filtering. e.g., for doors and windows.
// Remember for component family, you will need to check element type
and category.

var doorTypeCollector = new FilteredElementCollector(_doc);
doorTypeCollector.OfClass(typeof(FamilySymbol));
doorTypeCollector.OfCategory(BuiltInCategory.OST_Doors);
IList<Element> doorTypes = doorTypeCollector.ToElements();

ShowElementList(doorTypes, "Door Types (by Filter): ");
}

/// <summary>
/// Helper function to display info from a list of elements passed onto.
/// </summary>
public void ShowElementList(IList<Element> elems, string header)
{
    string s = " - Class - Category - Name (or Family: Type Name) - Id -
\r\n";
    foreach (Element e in elems)
    {
        s += ElementToString(e);
    }
    TaskDialog.Show(header + "(" + elems.Count.ToString() + "):", s);
}

/// <summary>
/// Helper function: summarize an element information as a line of text,
/// which is composed of: class, category, name and id.
/// name will be "Family: Type" if a given element is ElementType.
/// Intended for quick viewing of list of element, for example.
/// </summary>
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.Value.ToString() + "\r\n";
}
```

</C#>

これをテストするために、Execute() メソッドから ListFamilyTypes() を呼び出すことができます。

**議論:**

- 壁、床およびドア以外のファミリ タイプにはどんなものがありますか？
- それらを取得するためにどのメソッドを使用することができますか？

**実習:**

- オブジェクトのクラスを 1 つ選んで、そのファミリタイプをすべて取得するコードを記述してください。

### 3. 特定オブジェクト クラスのインスタンスをリストする

特定のオブジェクト タイプのインスタンスの一覧を取得するためには、フィルタを使用する必要があります。ファミリタイプで学習したものと同じ考え方が、インスタンスにも当てはまります。

下記は、壁インスタンスをすべて収集する例です:

```
<C#>
var wallCollector = new
    FilteredElementCollector(m_rvtDoc).OfClass(typeof(Wall));
IEnumerable<Element> wallList = wallCollector.ToElements();
</C#>
```

次の例は、ドア インスタンスをすべて収集するものです。

```
<C#>
var doorCollector = new
    FilteredElementCollector(m_rvtDoc).OfClass(typeof(FamilyInstance));
```



```

doorCollector.OfCategory(BuiltInCategory.OST_Doors);
IList<Element> doorList = doorCollector.ToElements();
</C#>

```

#### 議論:

- 壁、床およびドア以外のインスタンスにはどんなものがありますか？
- それらを取得するためにどのメソッドを使用することができますか？

#### 実習:

- 窓インスタンスをすべて取得するコードを記述してください。

## 4. 特定のファミリタイプを見つける

このセクションでは、特定のファミリタイプを見つける方法を学習します。次の要素を取得したいとしましょう:

- 壁タイプ — “標準壁: 一般 -200mm”
- ドアタイプ — “片側フラッシュ: 0915 x 2134mm”

### 4.1 特定の名前(文字列)と一致する壁タイプを見つける

壁タイプを取得するには、いくつかの異なる方法があります。まずは、LINQクエリを使用する方法から解説します:

```

<C#>
// Find a specific family type for a wall with a given family and type
// names. This version uses LINQ query.

public Element FindFamilyType_Wall_v1(
    string wallFamilyName,
    string wallTypeName)
{
    // narrow down a collector with class.

    var wallTypeCollector1 = new FilteredElementCollector(m_rvtDoc);
    wallTypeCollector1.OfClass(typeof(WallType));

    // LINQ query

    var wallTypeElems1 =
        from element in wallTypeCollector1
        where element.Name.Equals(wallTypeName)
        select element;

    // get the result.

```

```

    Element wallType1 = null; // result will go here.

    // (1) directly accessing from the query result.

    if (wallTypeElems1.Count<Element>() > 0)
    {
        wallType1 = wallTypeElems1.First<Element>();
    }

    // (2) If you want to get the result as a list of element, here is how.

    //IList<Element> wallTypeList1 = wallTypeElems1.ToList();
    //if (wallTypeList1.Count > 0)
    // wallType1 = wallTypeList1[0]; // Found it.

    return wallType1;
}
</C#>

```

次はイテレータを使用する方法です:

```

<C#>
// Find a specific family type for a wall, which is a system family.
// This version uses iteration. (cf. Developer guide 89)
//
public Element FindFamilyType_Wall_v2(
    string wallFamilyName,
    string wallTypeName)
{
    // first, narrow down the collector by Class
    var wallTypeCollector2 =
        new FilteredElementCollector(m_rvtDoc).OfClass(typeof(WallType));

    // use iterator
    FilteredElementIterator wallTypeItr =
        wallTypeCollector2.GetElementIterator();
    wallTypeItr.Reset();

    Element wallType2 = null;

    while (wallTypeItr.MoveNext())
    {
        WallType wType = (WallType)wallTypeItr.Current;
        // we check two names for the match: type name and family name.
        if ((wType.Name == wallTypeName)
&      (wType.get_Parameter(BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM).AsString
().Equals(wallFamilyName)))
        {
            wallType2 = wType; // we found it.
            break;
        }
    }

    return wallType2;
}

```

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

## 4.2 特定の名前(文字列)と一致する一致するドア タイプを見つける

同様に、ドア タイプについても異なる方法のアプローチが存在します。まずは、LINQ クエリを使用する方法を解説します:

```
<C#>  
// Find a specific family type for a door, which is a component family.  
// This version uses LINQ.  
  
public Element FindFamilyType_Door_v1(  
    string doorFamilyName,  
    string doorTypeName)  
{  
    // narrow down the collection with class and category.  
  
    var doorFamilyCollector1 = new FilteredElementCollector(m_rvtDoc);  
    doorFamilyCollector1.OfClass(typeof(FamilySymbol));  
    doorFamilyCollector1.OfCategory(BuiltInCategory.OST_Doors);  
  
    // parse the collection for the given name  
    // using LINQ query here.  
  
    var doorTypeElems =  
        from element in doorFamilyCollector1  
        where element.Name.Equals(doorTypeName) &&  
            element.get_Parameter(  
                BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM).  
                AsString().Equals(doorFamilyName)  
        select element;  
  
    // get the result.  
  
    Element doorType1 = null;  
  
    // (1) directly accessing from the query result  
    // we should have only one with the given name. minimum error checking.  
  
    //if (doorTypeElems.Count > 0)  
    // doorType1 = doorTypeElems(0) // found it.  
  
    // (2) if we want to get the list of element, here is how.  
  
    IList<Element> doorTypeList = doorTypeElems.ToList();  
    if (doorTypeList.Count > 0)  
    {  
        // we should have only one with the given name.  
        // minimum error checking  
        // found it.
```

```

        doorType1 = doorTypeList[0];
    }

    return doorType1;
}
</C#>

```

もう一方のアプローチは、ファミリからファミリ名を参照して、次に Family.GetFamilySymbolIds プロパティからのタイプ名を調べる方法です。

```

<C#>

    /// <summary>
    /// Find a specific family type for a door.
    /// another approach will be to look up from Family, then from Family.Sym
    bols property.
    /// This gets more complicated although it is logical approach.
    /// </summary>
    public Element FindFamilyType_Door_v2(string doorFamilyName, string doorT
    ypeName)
    {
        // (1) find the family with the given name.

        var familyCollector = new FilteredElementCollector(_doc);
        familyCollector.OfClass(typeof(Family));

        // Use the iterator
        Family doorFamily = null;
        FilteredElementIterator familyItr = familyCollector.GetElementIterator(
    );
        //familyItr.Reset();
        while ((familyItr.MoveNext()))
        {
            Family fam = (Family)familyItr.Current;
            // Check name and category
            if ((fam.Name == doorFamilyName) & (fam.FamilyCategory.Id.Value == (i
    nt)BuiltInCategory.OST_Doors))
            {
                // We found the family.
                doorFamily = fam;
                break;
            }
        }

        // (2) Find the type with the given name.

        Element doorType2 = null;
        // Id of door type we are looking for.
        if (doorFamily != null)
        {

```

```

        //FamilySymbolSet doorFamilySymbolSet = doorFamily.Symbols;           //
'Autodesk.Revit.DB.Family.Symbols' is obsolete:
        // 'This property is obsolete in Revit 2015. Use
Family.GetFamilySymbolIds() instead.'

        // Iterate through the set of family symbols.
        //FamilySymbolSetIterator doorTypeItr =
doorFamilySymbolSet.ForwardIterator();
        //while (doorTypeItr.MoveNext())
        //{
        //    FamilySymbol dType = (FamilySymbol)doorTypeItr.Current;
        //    if ((dType.Name == doorTypeName))
        //    {
        //        doorType2 = dType;    // Found it.
        //        break;
        //    }
        //}

        /// following part is modified code for Revit 2015

ISet<ElementId> familySymbolIds = doorFamily.GetFamilySymbolIds();

if (familySymbolIds.Count > 0)
{
    // Get family symbols which is contained in this family
    foreach (ElementId id in familySymbolIds)
    {
        FamilySymbol dType = doorFamily.Document.GetElement(id) as Family
Symbol;
        if ((dType.Name == doorTypeName))
        {
            doorType2 = dType;    // Found it.
            break;
        }
    }
}

return doorType2;
}
}
</C#>

```

#### 4.3 より汎用的な機能を定義する

これまでは、個別の用途に向けたフィルタを定義してきました。与えられたファミリ名やタイプ名の要素を取得する機能の中で、より汎用的な関数を作成すれば、さらに便利になります。下記の関数は、引数としてドキュメント、ファミリ名、タイプ名とオプションとなるカテゴリ情報を取り、ドキュメントで見つかったファミリ タイプを返します:

```

<C#>
// find an element of the given type, name, and category(optional)

public static Element FindFamilyType(
    Document rvtDoc, Type targetType,
    string targetFamilyName,
    string targetTypeNames,
    Nullable<BuiltInCategory> targetCategory)
{
    // first, narrow down to the elements of the given type and category

    var collector =
        new FilteredElementCollector(rvtDoc).OfClass(targetType);
    if (targetCategory.HasValue)
    {
        collector.OfCategory(targetCategory.Value);
    }

    // parse the collection for the given names
    // using LINQ query here.

    var targetElems =
        from element in collector
        where element.Name.Equals(targetTypeNames) &&
            element.get_Parameter(BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM).
                AsString().Equals(targetFamilyName)
        select element;

    // put the result as a list of element for accessibility.

    IList<Element> elems = targetElems.ToList();

    // return the result.

    if (elems.Count > 0)
    {
        return elems[0];
    }

    return null;
}
</C#>

```

この関数を使用すると、与えられた名前を持ったファミリタイプを以下のように見つけ出すことができます:

```

<C#>
ElementType wallType3 =
    (ElementType)FindFamilyType(m_rvtDoc, typeof(WallType),
        "標準壁", "一般 -200mm", null);

ElementType doorType3 =
    (ElementType)FindFamilyType(m_rvtDoc, typeof(FamilySymbol),

```

```
"片側フラッシュ", "0915 x 2134mm", BuiltInCategory.OST_Doors);  
</C#>
```

#### 実習:

- 与えられた名前のファミリ タイプを取得してファミリ タイプを返す FindFamilyType() を実装してください
- FindFamilyType() を使用して、壁タイプ、ドア タイプと窓タイプから好みのものを取得してください (ファミリ名はハードコードできます)

## 5. 特定インスタンスを見つける

### 5.1 特定のファミリ タイプに該当するインスタンスを見つける

他の状況では、特定のファミリ タイプに該当するインスタンスを取得したい場面があるかもしれません。下記の関数は、クラス タイプとファミリ タイプの要素 id、オプションとなるカテゴリをとり、与えられたファミリ タイプのインスタンスのリストを返します:

```
<C#>  
  
// find a list of element with given class, family type and  
// category (optional).  
public IList<Element> FindInstancesOfType(  
    Type targetType,  
    ElementId idType,  
    Nullable<BuiltInCategory> targetCategory)  
{  
    // narrow down to the elements of the given type and category  
  
    var collector =  
        new FilteredElementCollector(m_rvtDoc).OfClass(targetType);  
    if (targetCategory.HasValue)  
    {  
        collector.OfCategory(targetCategory.Value);  
    }  
  
    // parse the collection for the given family type id.  
    // using LINQ query here.  
  
    var elems =  
        from element in collector  
        where element.get_Parameter(BuiltInParameter.SYMBOL_ID_PARAM).  
            AsElementId().Equals(idType)
```

```

        select element;

        // put the result as a list of element fo accessibility.

        return elems.ToList();
    }
</C#>

```

例えば、この関数を使用すると、引数で渡したファミリタイプのインスタンスのリストを、次のように見つけることができます:

```

<C#>
IList<Element> walls = FindInstancesOfType(typeof(Wall), idWallType, null);
IList<Element> doors = FindInstancesOfType(typeof(FamilyInstance),
    idDoorType, BuiltInCategory.OST_Doors);
</C#>

```

## 5.2 与えられたクラスと名前で要素を見つける

一般に用いられている別のシナリオは、レベルやビューのような、Revit がサポートする要素を取得することかもしれません。次の関数は、各レベル要素などを取得するのに便利な関数になるでしょう。

```

<C#>

// find a list of element with given Class, Name and Category (optional).

public static IList<Element> FindElements(
    Document rvtDoc,
    Type targetType,
    string targetName,
    Nullable<BuiltInCategory> targetCategory)
{
    // first, narrow down to the elements of the given type and category

    var collector =
        new FilteredElementCollector(rvtDoc).OfClass(targetType);
    if (targetCategory.HasValue)
    {
        collector.OfCategory(targetCategory.Value);
    }

    // parse the collection for the given names
    // using LINQ query here.

```



```

var elems =
    from element in collector
    where element.Name.Equals(targetName)
    select element;

// put the result as a list of element for accessibility.

return elems.ToList();
}

// -----
// Helper function: searches elements with given Class, Name and
// Category (optional),
// and returns the first in the elements found.
// This gets handy when trying to find, for example, Level.
// e.g., FindElement(m_rvtDoc, GetType(Level), "Level 1")

public static Element FindElement(Document rvtDoc, Type targetType,
    string targetName, Nullable<BuiltInCategory> targetCategory)
{
    // find a list of elements using the overloaded method.
    IList<Element> elems =
        FindElements(rvtDoc, targetType, targetName, targetCategory);

    // return the first one from the result.
    if (elems.Count > 0)
    {
        return elems[0];
    }

    return null;
}
</C#>

```

例えば、この関数を使用すると、引数にファミリ タイプを渡して、インスタンスのリストを、次のように見つけることができます:

```

<C#>
Level level1 = (Level)FindElement(m_rvtDoc, typeof(Level), "レベル 1 ", null);
</C#>

```

後の実習では、単純な家を作成するためにこの関数を使用します。

## 実習:

- 引数としてドキュメント、クラス、名前およびオプションのカテゴリをとり、与えられたクラス、名前およびカテゴリで要素のリストを返す FindElements() 関数を実装する
- FindElements() を呼び出して、リスト内の最初の要素だけを返す FindElement() を実装する
- FindElement()を使用して、与えられた名前のレベル要素を取得する(レベル名はハードコード可能)

## 5.3 パラメータ(オプション)でフィルタリングする

ここまでで、要素をフィルタリングする基礎を習得しました。具体的には、次のクラスを使用する方法を学習しました:

- FilteredElementCollector
- ElementClassFilter
- ElementCategoryFilter

さらに異なる種類のフィルタが存在します。例えば:

- BoundingBoxContainsPointFilter
- ElementDesignOptionFilter
- ElementIsCurveDrivenFilter
- ElementIsElementTypeFilter
- ElementParameterFilter
- ...

Revit API 開発者用ガイドの[「フィルタリング」](#)項目では、フィルタリングについて記述しています。詳細については、このドキュメントを参照してください。

次のコードは、パラメータ フィルタの1例です。このコードは、次のように、評価する長さと同等の壁パラメータ値をチェックするパラメータ フィルタを使用しています。

```
wall.parameter(length) > 60 フィート
```

**重要:** Revit API で使われる長さの単位はフィートです。

<C#>

```
//  
// Optional - example of parameter filter.  
// find walls whose length is longer than a certain length. e.g., 60 feet  
//     wall.parameter(length) > 60 feet  
// This could get more complex than looping through in terms of  
// writing a code. See page 87 of Developer guide.  
  
public IList<Element> FindLongWalls()
```

```

{
    // constant for this function.
    const double kWallLength = 60.0;
    // 60 feet. hard coding for simplicity.

    // first, narrow down to the elements of the given type and category
    var collector =
        new FilteredElementCollector(m_rvtDoc).OfClass(typeof(Wall));

    // define a filter by parameter
    // 1st arg - value provider
    BuiltInParameter lengthParam = BuiltInParameter.CURVE_ELEM_LENGTH;
    long iLengthParam = (long)lengthParam;
    var paramValueProvider =
        new ParameterValueProvider(new ElementId(iLengthParam));

    // 2nd - evaluator
    FilterNumericGreater evaluator = new FilterNumericGreater();

    // 3rd - rule value
    double ruleVal = kWallLength;

    // 4th - epsilon
    const double eps = 1E-06;

    // define a rule
    var filterRule =
        new FilterDoubleRule(paramValueProvider, evaluator, ruleVal, eps);

    // create a new filter
    var paramFilter = new ElementParameterFilter(filterRule);

    // go through the filter
    IList<Element> elems = collector.WherePasses(paramFilter).ToElements();

    return elems;
}
</C#>

```

## 6. サマリ

この実習では、要素のフィルタリング方法を学習しました。学習した項目は、次のとおりです。

- ファミリタイプをリストする
- 特定オブジェクトクラスのインスタンスをリストする

- 特定のファミリ タイプを見つける
- 特定のインスタンスを見つける

次の実習では、Revit 内の要素を修正する方法を学習します。