**Revit API Intro Labs**

**Lab3 – Element Filtering**

Created by M. Harada, July 2010  
Updated by DevTech AEC WG  
Last modified: 3/11/2015

**<VB.NET>**VB.NET Version**</VB.NET>**

**Objective:** In this lab, we will learn how to obtain the elements that you are interested in, using filtering mechanism in Revit API. We’ll learn how to:

* Retrieve family types
* Retrieve instances of a specific object class
* Find a specific family type
* Find specific instances

**Tasks:** We’ll write a command that accumulatively demonstrates various methods and approaches for filtering elements. Use this lab as an exercise to familiarize yourself with filtering.

1. List family types (e.g., wall types, floor types, and door types)
2. List instances of a specific type of objects (e.g., walls and doors)
3. Find a specific family type (e.g., “Basic Wall: Generic – 200mm” wall type, “M\_Single-Flush: 0915 x 2134mm” door type)
4. Find specific Instances (i.e., instances of “Basic Wall: Generic – 200mm” wall type, instances of “M\_Single-Flush: 0915 x 2134mm” door type, and walls that are longer than a certain length.)

Figure 1 shows the sample images of output after running the command that you will be defining in this lab:

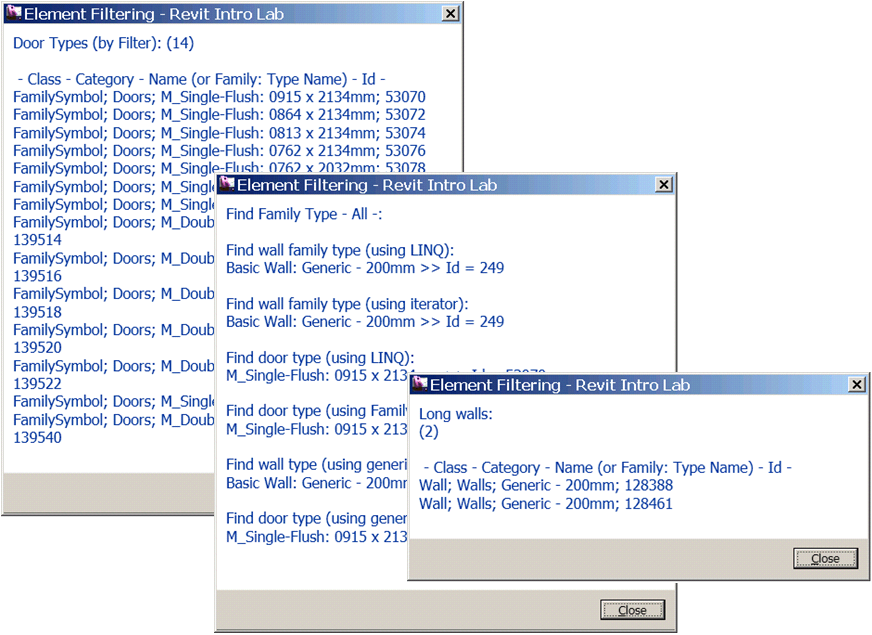


Figure 1. Dialogs showing the result of various filtering

The following is the breakdown of step by step instructions in this lab:

1. Define a New External Command
2. List Family Types
3. List Instances of Specific Object Class
4. Find a Specific Family Type
5. Find Specific Instances
6. Summary
7. **Define A New External Command**

We’ll add another external command to the current project.

* 1. Add a new file and define another external command to your project. Let’s name them as follows:
* File name: **3\_ElementFiltering.vb (or .cs)**
* Command class name: **ElementFiltering**

(Once again, you may choose to use any names you want here. When you do so, just remember what you are calling your own project, and substitute these names as needed while following the instruction in this document.)

**Required Namespaces:**

Namespaces needed for this lab are:

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

Note (VB.NET only): if you are writing in VB.NET and you import namespaces at the project level, (i.e., in the project properties, there is no need to explicitly import in each file.

* 1. Like we did in Lab2, define member variables, e.g., m\_rvtApp and m\_rvtDoc, to keep DB level application and document respectively. The following is an example:

**<VB.NET>**

'' Element Filtering – learn about Revit element filtering

<Transaction(TransactionMode.Manual)> \_

Public Class ElementFiltering

Implements IExternalCommand

**'' member variables**

**Dim m\_rvtApp As Application**

**Dim m\_rvtDoc As Document**

Public Function Execute(ByVal commandData As ExternalCommandData, \_

ByRef message As String, \_

ByVal elements As ElementSet) \_

As Result \_

Implements IExternalCommand.Execute

'' Get the access to the top most objects.

**Dim rvtUIApp As UIApplication = commandData.Application**

**Dim rvtUIDoc As UIDocument = rvtUIApp.ActiveUIDocument**

**m\_rvtApp = rvtUIApp.Application**

**m\_rvtDoc = rvtUIDoc.Document**

'' ...

Return Result.Succeeded

End Function

End Class  
**</VB.NET>**

1. **List Family Types**

In the previous lab, we have learned that depending on whether the element is component family-based or system family, we will need to take a different approach to identify an element, using class names and categories. When we are retrieving the list of family types stored in the project database, the similar rules applies.

2.1 System Family Types

You can think of Revit elements as a bundle in a large sack and it is in a database. To access elements in it, you will need to query for it. As an example, the following will collect all the WallType class in the document:

**<VB.NET>**

Dim wallTypeCollector1 = New FilteredElementCollector(m\_rvtDoc)

wallTypeCollector1.WherePasses( \_

New ElementClassFilter(GetType(WallType)))

Dim wallTypes1 As IList(Of Element) = wallTypeCollector1.ToElements

**</VB.NET>**

FilteredElementCollector is a “container” object to collect elements which we are interested in. We first create it. And it passes through a filter, in this case, a class filter to filter out only to collect elements whose class is WallTypes. The last line converts a filtered element collector into a list of element; this is for convenience for further handling.

In our context, using filtering to retrieve a list of wall types may not be seen as a valuable thing to do. But this may become more convenient if your query becomes more complex.

Revit API offers various alternative forms for filtering for convenience. The following are the same as first two lines in the above code, but using OfClass():

**<VB.NET>**

Dim wallTypeCollector2 = New FilteredElementCollector(m\_rvtDoc)

wallTypeCollector2.OfClass(GetType(WallType))

**</VB.NET>**

You can further simplify it using shortcut:

**<VB.NET>**

Dim wallTypeCollector3 = \_

New FilteredElementCollector(m\_rvtDoc).OfClass(GetType(WallType))

**</VB.NET>**

2.3 Component Family Types

For component family, you will need to check element class and categories. Following is an example of getting a list of door family types.

**<VB.NET>**

Dim doorTypeCollector = New FilteredElementCollector(m\_rvtDoc)

doorTypeCollector.OfClass(GetType(FamilySymbol))

doorTypeCollector.OfCategory(BuiltInCategory.OST\_Doors)

Dim doorTypes As IList(Of Element) = doorTypeCollector.ToElements

**</VB.NET>**

The following code demonstrates the usage:

**<VB.NET>** Public Sub ListFamilyTypes()

''

'' (1) get a list of family types available in the current

'' rvt project.

'' (1.1a) here is an example with wall type.

Dim wallTypeCollector1 = New FilteredElementCollector(m\_rvtDoc)

wallTypeCollector1.WherePasses( \_

New ElementClassFilter(GetType(WallType)))

Dim wallTypes1 As IList(Of Element) = wallTypeCollector1.ToElements

'' using a helper funtion to display the result here. See code below.

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

'' (1.1b) the following are the same as two lines above.

'' these alternative forms are provided for convenience.

'' using OfClass()

''

'Dim wallTypeCollector2 = New FilteredElementCollector(m\_rvtDoc)

'wallTypeCollector2.OfClass(GetType(WallType))

'' (1.1c) the following are the same as above. For convenience.

'' using short cut this time.

''

'Dim wallTypeCollector3 = \_

' New FilteredElementCollector(m\_rvtDoc).OfClass(GetType(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.

'' for example, doors and windows.

'' remember for component family, you will need to check

'' element type and category

Dim doorTypeCollector = New FilteredElementCollector(m\_rvtDoc)

doorTypeCollector.OfClass(GetType(FamilySymbol))

doorTypeCollector.OfCategory(BuiltInCategory.OST\_Doors)

Dim doorTypes As IList(Of Element) = doorTypeCollector.ToElements

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

End Sub

'' Helper function to display info from a list of elements passed onto.

Sub ShowElementList( \_

ByVal elems As IList(Of Element), ByVal header As String)

Dim s As String = String.Empty

s += " - Class - Category - Name (or Family: Type Name) - Id - " + vbCr

For Each e As Element In elems

s += ElementToString(e)

Next

TaskDialog.Show(header + "(" + elems.Count.ToString() + "):", s)

End Sub

'' 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.

Function ElementToString(ByVal e As Element) As String

If e Is Nothing Then

Return "none"

End If

Dim name As String = ""

If TypeOf e Is ElementType Then

Dim param As Parameter = \_

e.Parameter(BuiltInParameter.SYMBOL\_FAMILY\_AND\_TYPE\_NAMES\_PARAM)

If param IsNot Nothing Then

name = param.AsString

End If

Else

name = e.Name

End If

Return e.GetType.Name + "; " + e.Category.Name + "; " \_

+ name + "; " + e.Id.IntegerValue.ToString + vbCr

End Function

**</VB.NET>**

To test this, you can call ListFamilyTypes() from the main Execute() method.

**Discussion:**

* Give examples of family types other than walls, floor and doors.
* Which methods can we use to retrieve them?

**Exercise:**

* Choose one class of object, write a code to retrieve all its family types.

1. **List Instances of a Specific Object Class**

To get a list of instances of specific object type, you will need to use filters. The same idea that we learned for family types applies for instances as well.

Here is an example of collecting all the wall instances:

**<VB.NET>**

Dim wallCollector = \_

New FilteredElementCollector(m\_rvtDoc).OfClass(GetType(Wall))

Dim wallList As IList(Of Element) = wallCollector.ToElements

**</VB.NET>**

Here is another for collecting all the doors.

**<VB.NET>**

Dim doorCollector = New FilteredElementCollector(m\_rvtDoc). \_

OfClass(GetType(FamilyInstance))

doorCollector.OfCategory(BuiltInCategory.OST\_Doors)

Dim doorList As IList(Of Element) = doorCollector.ToElements

**</VB.NET>**

**Discussion:**

* Give examples of instances other than walls, floor and doors.
* Which methods can we use to retrieve them?

**Exercise:**

* Write a code to retrieve all the instances of windows (or your choice.)

1. **Find a Specific Family Type**

In this section, we will look at the way to find a specific family type. Let’s say, we would like to retrieve:

* wall type - “Basic Wall: Generic – 200mm”
* door type – “M\_Single-Flush: 0915 x 2134mm”
  1. Find a wall type with a given name.

Let’s start with the wall. There are a few different ways to do this. The first version is to use LINQ query.

**<VB.NET>**

'' Find a specific family type for a wall with a given family and type

'' name. This version uses LINQ query.

Function FindFamilyType\_Wall\_v1( \_

ByVal wallFamilyName As String, \_

ByVal wallTypeName As String) As Element

'' narrow down a collector with class.

Dim wallTypeCollector1 = New FilteredElementCollector(m\_rvtDoc)

wallTypeCollector1.OfClass(GetType(WallType))

'' LINQ query

Dim wallTypeElems1 = \_

From element In wallTypeCollector1 \_

Where element.Name.Equals(wallTypeName) \_

Select element

'' get the result.

Dim wallType1 As Element = Nothing '' result will go here.

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

If wallTypeElems1.Count > 0 Then

wallType1 = wallTypeElems1.First

End If

'' (2) if you want to get the result as a list of element,

'' here is how.

'Dim wallTypeList1 As IList(Of Element) = wallTypeElems1.ToList()

'If wallTypeList1.Count > 0 Then

' wallType1 = wallTypeList1(0) ' found it.

'End If

Return wallType1

End Function

**</VB.NET>**

The second version uses iterations:

**<VB.NET>**

'' Find a specific family type for a wall, which is a system family.

'' This version uses iteration. (cf. Developer guide 89)

Function FindFamilyType\_Wall\_v2( \_

ByVal wallFamilyName As String, \_

ByVal wallTypeName As String) As Element

'' first, narrow down the collector by Class

Dim wallTypeCollector2 = \_

New FilteredElementCollector(m\_rvtDoc).OfClass(GetType(WallType))

'' use iterator

Dim wallTypeItr As FilteredElementIterator = \_

wallTypeCollector2.GetElementIterator

wallTypeItr.Reset()

Dim wallType2 As Element = Nothing

While wallTypeItr.MoveNext

Dim wType As WallType = wallTypeItr.Current

'' we check two names for the match: type name and family name.

If (wType.Name = wallTypeName) And \_

(wType.Parameter(BuiltInParameter.SYMBOL\_FAMILY\_NAME\_PARAM). \_  
 AsString.Equals(wallFamilyName)) Then

wallType2 = wType '' we found it.

Exit While

End If

End While

Return wallType2

End Function

**</VB.NET>**

4.2 Finding a door type with a given name

Similarly, for door types, you can also approach different ways. The first version uses LINQ query:

**<VB.NET>**

'' Find a specific family type for a door, which is a component family.

'' This version uses LINQ.

''

Function FindFamilyType\_Door\_v1(ByVal doorFamilyName As String, ByVal doorTypeName As String) As Element

'' narrow down the collection with class and category.

Dim doorFamilyCollector1 = New FilteredElementCollector(m\_rvtDoc)

doorFamilyCollector1.OfClass(GetType(FamilySymbol))

doorFamilyCollector1.OfCategory(BuiltInCategory.OST\_Doors)

'' parse the collection for the given name

'' using LINQ query here.

Dim doorTypeElems = \_

From element In doorFamilyCollector1 \_

Where element.Name.Equals(doorTypeName) And \_

element.Parameter(BuiltInParameter.SYMBOL\_FAMILY\_NAME\_PARAM).AsString.Equals(doorFamilyName) \_

Select element

'' get the result.

Dim doorType1 As Element = Nothing

'' (1) directly accessing from the query result

'If doorTypeElems.Count > 0 Then '' we should have only one with the given name. minimum error checking.

' doorType1 = doorTypeElems(0) ' found it.

'End If

'' (2) if we want to get the list of element, here is how.

Dim doorTypeList As IList(Of Element) = doorTypeElems.ToList()

If doorTypeList.Count > 0 Then '' we should have only one with the given name. minimum error checking.

doorType1 = doorTypeList(0) ' found it.

End If

Return doorType1

End Function

**</VB.NET>**

Another approach will be to look up a family name from Family, then the type name from Family.Symbols property. Although this is a logical approach, it looks more complex:

**<VB.NET>**

'' Find a specific family type for a door.

'' Look up from Family, then from Family.Symbols property.

Function FindFamilyType\_Door\_v2(ByVal doorFamilyName As String, ByVal doorTypeName As String) As Element

'' (1) find the family with the given name.

Dim familyCollector = New FilteredElementCollector(m\_rvtDoc)

familyCollector.OfClass(GetType(Family))

'' use the iterator

Dim doorFamily As Family = Nothing

Dim familyItr As FilteredElementIterator = \_

familyCollector.GetElementIterator

'familyItr.Reset()

While (familyItr.MoveNext)

Dim fam As Family = familyItr.Current

'' check name and categoty

If (fam.Name = doorFamilyName) And \_

(fam.FamilyCategory.Id.IntegerValue = BuiltInCategory.OST\_Doors) Then

'' we found the family.

doorFamily = fam

Exit While

End If

End While

'' (2) find the type with the given name.

''

Dim doorType2 As Element = Nothing '' id of door type we are looking for.

If doorFamily IsNot Nothing Then

'' if we have a family, then proceed with finding a type under

'' Symbols property.

Dim doorFamilySymbolIds As ISet(Of ElementId) =

doorFamily.GetFamilySymbolIds()

For Each id As ElementId In doorFamilySymbolIds

Dim dType As FamilySymbol =

TryCast(doorFamily.Document.GetElement(id), FamilySymbol)

If (dType.Name = doorTypeName) Then

doorType2 = dType ' Found it

Exit For

End If

Next

End If

Return doorType2

End Function

**</VB.NET>**

4.3 Defining a more generalized function

So far, we have defined a filter for individual cases. Sometimes having a more generalized form of function to retrieve an element of a given family and type name may get handy. The following function takes a document, the name of family, the name of type, and optional category information as arguments, and returns the family type found in the document:

**<VB.NET>**

'' Find an element of the given type, name, and ategory(optional).

Public Shared Function FindFamilyType( \_

ByVal rvtDoc As Document, ByVal targetType As Type, \_

ByVal targetFamilyName As String, \_

ByVal targetTypeName As String, \_

Optional ByVal targetCategory As BuiltInCategory = Nothing) \_

As Element

'' first, narrow down to the elements of the given type and category

Dim collector = \_

New FilteredElementCollector(rvtDoc).OfClass(targetType)

If Not (targetCategory = Nothing) Then

collector.OfCategory(targetCategory)

End If

'' parse the collection for the given names

'' using LINQ query here.

Dim targetElems = \_

From element In collector \_

Where element.Name.Equals(targetTypeName) And \_

element.Parameter(BuiltInParameter.SYMBOL\_FAMILY\_NAME\_PARAM). \_

AsString.Equals(targetFamilyName) \_

Select element

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

Dim elems As IList(Of Element) = targetElems.ToList()

'' return the result.

If elems.Count > 0 Then

Return elems(0)

End If

Return Nothing

End Function

**</VB.NET>**

Using this function, you can find a family type with a given name as follows, e.g.,:

**<VB.NET>**

Dim wallType3 As Element = \_  
 FindFamilyType(m\_rvtDoc, GetType(WallType), \_

"Basic Wall", "Generic - 200mm")

Dim doorType3 As Element = \_

FindFamilyType(m\_rvtDoc, GetType(FamilySymbol), \_

"M\_Single-Flush", "0915 x 2134mm", BuiltInCategory.OST\_Doors)

**</VB.NET>**

**Exercise:**

* Implement FindFamilyType() that retrieves a family type of given name and return the family type.
* Using FindFamilyType(), retrieve a wall, door and window type of your choice. (You can hard code the family names.)

1. **Find Specific Instances**

5.1 Find Instances of a given family type

Another situation might be that we want to retrieve instances of a given family type. The following function takes a class, the element id of a certain family type, and optional category, and returns a list of elements that are instance of the given family type:

**<VB.NET>**

'' Find a list of element with the given Class, family type and

'' Category (optional).

Function FindInstancesOfType( \_

ByVal targetType As Type, \_

ByVal idType As ElementId, \_

Optional ByVal targetCategory As BuiltInCategory = Nothing) \_  
 As IList(Of Element)

'' narrow down to the elements of the given type and category

Dim collector = \_

New FilteredElementCollector(m\_rvtDoc).OfClass(targetType)

If Not (targetCategory = Nothing) Then

collector.OfCategory(targetCategory)

End If

'' parse the collection for the given family type id.

'' using LINQ query here.

Dim elems = \_

From element In collector \_

Where element.Parameter(BuiltInParameter.SYMBOL\_ID\_PARAM). \_

AsElementId.Equals(idType) \_

Select element

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

Return elems.ToList()

End Function

**</VB.NET>**

For example, using this function, you can find a list of instances of a given family type as follows:

**<VB.NET>**

Dim walls As IList(Of Element) = \_   
 FindInstancesOfType(GetType(Wall), idWallType)

Dim doors As IList(Of Element) = \_  
 FindInstancesOfType(GetType(FamilyInstance), idDoorType, \_   
 BuiltInCategory.OST\_Doors)

**</VB.NET>**

5.2 Find an element with a given class and a name

One other commonly used scenario would be to retrieve some supporting elements in Revit, such as Level and View element. The following functions will be another handy function to retrieve such as each Level element.

**<VB.NET>**

'' Find a list of elements with given class, name, category (optional).

Public Shared Function FindElements( \_

ByVal rvtDoc As Document, \_

ByVal targetType As Type, \_

ByVal targetName As String, \_

Optional ByVal targetCategory As BuiltInCategory = Nothing) \_   
 As IList(Of Element)

'' narrow down to the elements of the given type and category

Dim collector = \_   
 New FilteredElementCollector(rvtDoc).OfClass(targetType)

If Not (targetCategory = Nothing) Then

collector.OfCategory(targetCategory)

End If

'' parse the collection for the given names

'' using LINQ query here.

Dim 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()

End Function

'' -----------------------------------------------------------------

'' Searches elements with given Class, Name and Category (optional),

'' and returns the first in the elements found.

Public Shared Function FindElement(ByVal rvtDoc As Document, \_

ByVal targetType As Type, \_   
 ByVal targetName As String, \_

Optional ByVal targetCategory As BuiltInCategory = Nothing) \_

As Element

'' find a list of elements using the overloaded method.

Dim elems As IList(Of Element) = \_   
 FindElements(rvtDoc, targetType, targetName, targetCategory)

'' return the first one from the result.

If elems.Count > 0 Then

Return elems(0)

End If

Return Nothing

End Function

**</VB.NET>**

For example, using this function, you can find a list of instances of a given family type as follows:

**<VB.NET>**

Dim level1 As Level = FindElement(m\_rvtDoc, GetType(Level), "Level 1")

**</VB.NET>**

We’ll use this in the later labs when we want to create a simple house.

**Exercise:**

* Implement FindElements() function that takes a document, class, name and optional category as arguments, and returns a list of elements with the given class, name and category.
* Implement FindElement() that calls FindElements(), and returns only the first element in the list.
* Using FindElement(), retrieve a Level element of a given name (You can hard code the Level name)

5.3 Filtering with parameters (Optional)

By now, you must have familiarized yourself with the basics of the element filtering. More specifically, we have learned how to use the following classes:

* FilteredElementCollector
* ElementClassFilter
* ElemetCategoryFilter

There are more different kinds of filters, such as:

* BoundingBoxContainsPointFilter
* ElementDesignOptionFilter
* ElementIsCurveDrivenFilter
* ElementIsElementTypeFilter
* ElementParameterFilter
* …

[This section](http://www.autodesk.com/revit-help/?guid=GUID-85E4A43E-88B5-43C6-908C-2D138C9F611D) of the Revit developer documentation describes filtering. For more detail, please take a look at the documentation.

One thing to note is that a filter can be “Quick” or “Slow”. We have not discussed about in this scope of the first day API training labs. But this is something you may be aware of. When the performance becomes of concern, you should definitely take a look at how to filter elements. If needed, you may perform a performance test among possible approaches as well.

Below is one example of parameter filter. The code uses parameter filter to check the wall’s parameter value for length, which is equivalent of evaluating:

wall.parameter(length) > 60 feet

**<VB.NET>**

'' example of parameter filter.

'' find walls whose length is longer than 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.

Function FindLongWalls() As IList(Of Element)

'' constant for this function.

Const kWallLength As Double = 60.0 '' 60 feet. hard coding

'' first, narrow down to the elements of the given type and category

Dim collector = \_   
 New FilteredElementCollector(m\_rvtDoc).OfClass(GetType(Wall))

'' define a filter by parameter

'' 1st arg - value provider

Dim lengthParam As BuiltInParameter = \_   
 BuiltInParameter.CURVE\_ELEM\_LENGTH

Dim iLengthParam As Integer = lengthParam

Dim paramValueProvider = \_   
 New ParameterValueProvider(New ElementId(iLengthParam))

'' 2nd - evaluator

Dim evaluator As New FilterNumericGreater

'' 3rd - rule value

Dim ruleVal As Double = kWallLength

'' 4th - epsilon

Const eps As Double = 0.000001

'' define a rule

Dim filterRule = New FilterDoubleRule( \_   
 paramValueProvider, evaluator, ruleVal, eps)

'' create a new filter

Dim paramFilter = New ElementParameterFilter(filterRule)

'' go through the filter

Dim elems As IList(Of Element) = \_   
 collector.WherePasses(paramFilter).ToElements

Return elems

End Function

**</VB.NET>**

1. **Summary**

In this lab, we have learned how to filter elements. We have learned how to:

* Retrieve family types
* Retrieve instances of a specific object class
* Find a specific family type
* Find specific instances

In the next lab, we will take a look at how to modify elements in the Revit.

Autodesk Developer Network