```
using System;
using System.Collections.Generic;
using System.Text;

using Autodesk.Revit;
using Autodesk.Revit.Elements;
using Autodesk.Revit.Parameters;
using System.Windows.Forms;
using System.Diagnostics;

namespace ParameterUtil
{
    /// <summary>
    /// This class ParamUtil is used to retrieve the parameter set of a selected element
    /// </summary>
    class ParamUtil:IExternalCommand
    {
        public IExternalCommand.Result Execute(ExternalCommandData revit, ref String message, ElementSet elements)
        {
            // current active Revit document
            Document doc = revit.Application.ActiveDocument;

            // check if only one element is being selected
            if (doc.Selection.Elements.Size == 1)
            {

                // get the element iterator from the current selected object set
                ElementSetIterator iter = doc.Selection.Elements.ForwardIterator();

                // by using MoveNext() function to locate the first element of the list
                iter.MoveNext();
                Element sel = iter.Current as Element;

                // check if current selected element is not null
                if (sel != null)
                {
                    // initiate a StringBuilder object to record the parameter info
                    StringBuilder sb = new StringBuilder();

                    // append the element name at the beginning
                    sb.AppendFormat("\n--- Select Element:{0} ---\n", (sel.Name));

                    // iterate through each parameter that is associated with the selected element
                    foreach (Parameter p in sel.Parameters)
                    {
                        // if the parameter object is not null
                        if (p != null)
                        {
```

```
                        // append the name of current parameter
                        sb.AppendFormat("Name:[ {0} ], ", p.Definition.Name);

                        // append the type and value of the parameter
                        // 5 types of parameters: integer, double, ElementId, string, None
                        switch (p.StorageType)
                        {
                            case StorageType.Integer:
                                sb.AppendFormat(" Type:[ integer ], Value:[{0}]\n", p.AsInteger());
                                break;
                            case StorageType.Double:
                                sb.AppendFormat(" Type:[ double ], Value:[{0}]\n", p.AsDouble());
                                break;
                            case StorageType.ElementId:
                                // if the storage type is an ElementId
                                // use get_element(ref ElementId id) to retirve the reference to the element
                                ElementId eId = p.AsElementId();
                                Element e = doc.get_Element(ref eId);

                                // append the element name as the value
                                sb.AppendFormat(" Type:[ ElementId ], Value:[{0}]\n", (e != null ? e.Name : "Not set" ↙
    ));
                                break;
                            case StorageType.String:
                                sb.AppendFormat(" Type:[ string ], Value:[{0}]\n", p.AsString());
                                break;
                            case StorageType.None:
                                sb.AppendFormat(" Type:[ none ], Value:[{0}]\n", p.AsValueString());
                                break;
                            default:
                                MessageBox.Show("Can not match the default parameter types!\n");
                                break;
                        }
                    }
                }

                // add an ending statement at the end
                sb.AppendLine("End of parsing Parameter set!\t-------------\n");

                // out put the property string to the Immediate Window
                Debug.WriteLine(sb.ToString());
            }
            else
            {
                MessageBox.Show("iterator falls to collect info!");
                return IExternalCommand.Result.Failed;
            }
```

```
            }
            else
            {
                MessageBox.Show("you show select a element!");
                return IExternalCommand.Result.Failed;
            }


            return IExternalCommand.Result.Succeeded;

        }
    }
}
```