AUTODESK UNIVERSITY 2013

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

Angel Velez – Autodesk, Inc.

DV2020

Since the Autodesk Revit 2012 release, Industry Foundation Classes (IFC) export from Revit has been based on .NET open source code, enabling Revit API users who are familiar with IFC to make their own customized versions of both the exporter and the associated UI. This class covers the major design concepts for the export code with the intention of understanding how to modify the open source code or how to create a custom exporter on top of the existing exporter. This class is intended for advanced users who know the principals of API development and/or IFC, and it includes looking at Revit 2014 open source .NET code.

Learning Objectives

At the end of this class, you will be able to:

- Find, download, and build the Revit IFC Export open source code
- Describe the overall design of the Revit IFC Export open source code
- Make simple changes to the code, such as adding a property set
- Create a custom exporter on top of the open source code

About the Speaker

Angel Velez is a Senior Principal Engineer at Autodesk, Inc. He graduated from MIT with B.S. degrees in Computer Science and Mathematics in 1992, and Stanford University with a Master's degree in Computer Science in 1994. In 1999, after working in the mechanical CAD industry for 5 years, he joined Charles River Software to work on a project that eventually became known as Revit. Since 2004, Angel has been working primarily on interoperability issues, concentrating on IFC.

angel.velez@autodesk.com

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

Getting the code

There are two ways to get the source code for the IFC exporter. The best way is to download it via the source control system on SourceForge. To do this, you'll need to do the following steps:

1. Get a Subversion (SVN) client.

The easiest way to do this is to get TortoiseSVN from SourceForge. You can, however, use any SVN client you want. You'll need to install this client before you continue to the next step. The rest of the steps assume you are using TortoiseSVN; if not, there are similar steps that will be needed for your particular client.

2. Create a directory for your source code.

For this example below, I've created a directory called "TestSVN" on my desktop. The location of the directory is completely up to you.

3. Checkout the code.

Right-click on the directory created above, and you will get an item "SVN Checkout…" on the context-sensitive menu. Select that item, and you will get the window in Table 1 below.

Click "OK" to download the code.

Note that the above code is for the 2014 IFC exporter. You can also modify the 2013 exporter code, or the UI for 2013 or 2014. The directories for each are:

- 2014IFCExporterUI: Alternate UI for 2014.
- BIM.IFC: 2013 exporter
- IFCExporterUI: Alternate UI for 2013
- IFCExtension: 2013 bridge between alternate UI and exporter to allow setting extra options, used by both the 2013 exporter and the 2013 UI projects.

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

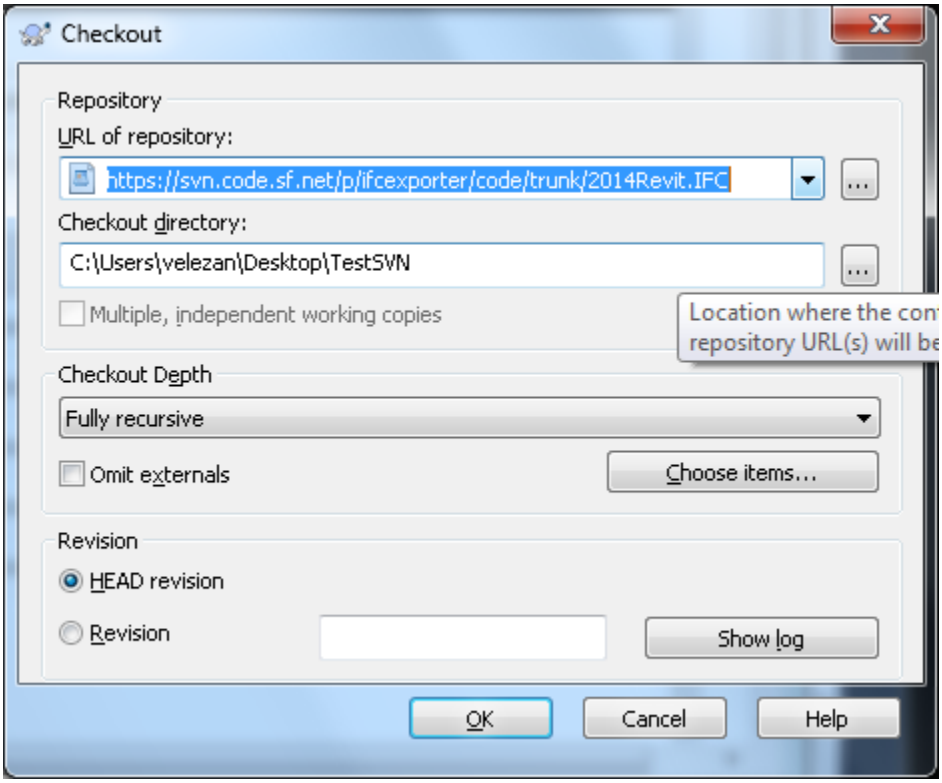


Table 1. SVN Checkout window.

4. Update the .props files.

There are three files that will need to be updated based on your specific installation of Revit. These are:

- Install/Modify Addin file.props
- Source/Revit.IFC.Common/Revit.IFC.Common.props
- Source/Revit.IFC.Export/Revit.IFC.Export.props

They will have corresponding files with names like: Revit.IFC.Common.props.template - copy this file and change its name to Revit.IFC.Common.props.

For Modify Addin file.props, simply delete the file and rename the corresponding template to Modify Addin file.props. For the other two files, open them in Notepad (or your favourite text editor), and find the two lines:

<!-- Change to your directory of Revit installation. -->

Below them, you will see something like:

<HintPath>C:\Program Files\Autodesk\Revit Architecture 2014\Program\RevitAPI.dll</HintPath>

Change that to the location of you Revit installation. After you have made these changes, delete the original .props file and rename the one you've just edited to .props.

5. Final clean up steps

There are two final minor steps required to build the solution. First, remove Revit.IFC.Export.sig from Revit.IFC Setup by right-clicking on the name in the Solution Explorer and choosing "Remove", as seen in Table 2 below. This file is required for LT support, but customizing of the Open Source exporter for Revit LT by third parties is not currently allowed.

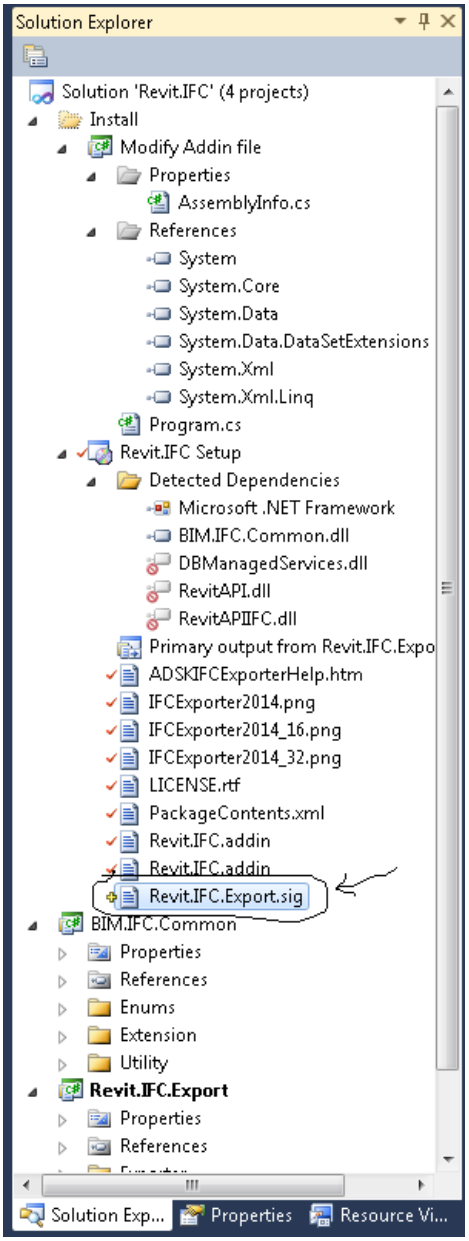


Table 2. Revit.IFC.Export.sig file.

Secondly, open Revit.IFC.Export.csproj in Notepad and delete:

```
  <PropertyGroup>

    <PostBuildEvent>call $(SolutionDir)SignFile.bat $(TargetPath)

$(SolutionDir)..\..\ThirdParty\RevitAPI\Identification\SignData\Release\SignData.exe $(TargetPath)
$(SolutionDir)..\..\ThirdParty\RevitAPI\Identification\pair.dat LT

perl $(SolutionDir)PostBuild.pl $(ProjectDir) $(TargetPath) $(Configuration) $(Platform)</PostBuildEvent>

  </PropertyGroup>
```

Although the solution will build with those lines, it will generate unnecessary errors.

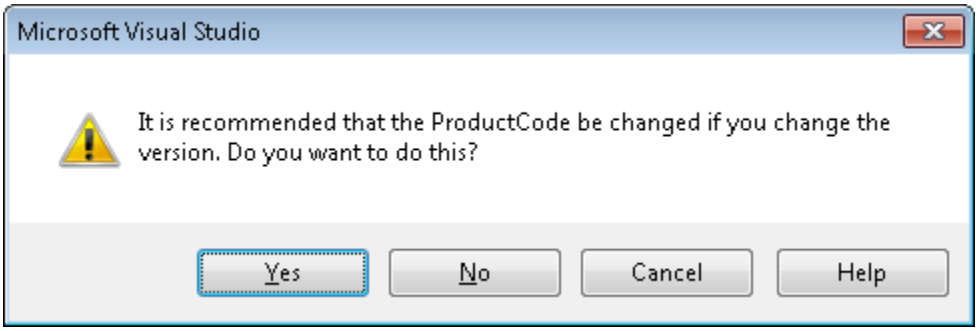
6. Build the solution

This should create the installer in either Install\Setup\Debug or Install\Setup\Release, depending on your settings. Note that you will then need to run the installer to update the export.

7. (Optional) Update the version

If you plan to work directly on the open source code, instead of creating a new exporter on top of it, you may want to increase the version number to convince the installer to overwrite your previous installed versions. To do this:

- Locate the three AssemblyInfo.cs files in the “Modify Addin file”, “BIM.IFC.Common” and “Revit.IFC.Export” projects.
- Update the lines below to your latest version. In general, the recommendation is to change the last two digits.
  ```
  [assembly: AssemblyVersion("3.7.1.0")]
  [assembly: AssemblyFileVersion("3.7.1.0")]
  ```
- Go to the “Properties” palette and update the “Version” property. You will get a property box like the one below: click “Yes”.



- Save all of the files, rebuild, and reinstall.

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

Overall Design of the Open Source Code

Top Level

The top level code resides entirely in Exporter.cs. The exporter is registered as an external application via IExternalDBApplication with the following code:

```
private void OnApplicationInitialized(object sender, EventArgs eventArgs)
{
  SingleServerService service =
ExternalServiceRegistry.GetService(ExternalServices.BuiltInExternalServices.IFCExporterService) as
SingleServerService;
  if (service != null)
  {
    Exporter exporter = new Exporter();
    service.AddServer(exporter);
    service.SetActiveServer(exporter.GetServerId());
  }
}
```

The Exporter class above needs to be of type IExporterIFC, and contains the implementation of the exporter. The entry point for the export is:

```
public void ExportIFC(Autodesk.Revit.DB.Document doc, ExporterIFC exporterIFC,
Autodesk.Revit.DB.View filterView)
```

"Doc" is the current document being exported. The "exporterIFC" argument (no relation to IExporterIFC) is initialized in native code, and allows interaction between native and .NET code, including providing access to the toolkit that reads and writes IFC file. The "filterView" argument is optional, and is used for "Current View Only" export.

In addition to the ExporterIFC class, the ExporterIFCUtils API allows access to utility functions in native code that aren't available in the "standard" Revit API. These include:

- Functions necessary for legacy element export,
- Element access functions needed for export not in standard API (although these may be deprecated in future as some are moved to general Revit API),
- Some routines not yet converted to .NET (these may also be deprecated in future releases, as the last remaining code is replaced with .NET code).

ExportIFC function

The ExportIFC function is very simple, and consists of three basic steps: BeginExport, element traversal, and EndExport. Each of these functions will be covered below.

BeginExport

The BeginExport does the initialization of the export, and creates the top-level entities necessary for the rest of export. This includes:

- Initializing IFCFile based on schema.
- Initializing property sets and quantities to use.
- Creating unique and top-level IFC entities, including IfcProject, IfcBuilding, and IfcBuildingStoreys.
- Creating commonly used directions and Cartesian points for re-use.

Element traversal

InitializeElementExporters creates the set of delegates that order the element traversal of the document. By default, elements are processed in the following order:

- Spatial elements (rooms, areas, MEP spaces)
- Most non-spatial elements (primarily, elements with 3D geometry)
- Containers. These are generally elements that are containers of other elements or entities, such as railings, fabric areas, trusses, beam systems, area schemes, zones
- Grids
- MEP connectors

Users that create their own exporter can override this order (see the final section on creating your own exporter for more information).

Each element is handled as generically as possible, and creates 1 or more IfcBuildingElement entities. In the cases where there is a 1-to-1 correspondence, we can generate a consistent GUID. In cases where there is a 1-to-many correspondence, we may need to generate some random GUIDs for each export. In some cases, IFC entities contain data relating one entity to one or more other entities. In the cases where it isn't known if all of the elements have already been created, export will cache the information for use in a later traversal stage, or in EndExport.

Here is a simple example of processing an element: exporting a Footing, from the open source code with extra annotation.

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

```
public static void ExportFooting(ExporterIFC exporterIFC, Element element,
  GeometryElement geometryElement,
  string ifcEnumType, ProductWrapper productWrapper)
{
  // We do several checks to see if an element is a Part.  This is the first check.
  bool exportParts = PartExporter.CanExportParts(element);
  if (exportParts && !PartExporter.CanExportElementInPartExport(element,
    element.LevelId, false))
    return;
  IFCFile file = exporterIFC.GetFile();
  // The top level IFCTransaction class is not a Revit Transaction, but instead allows
  // us to delete partially created IFC entities if the IfcFooting creation files.
  using (IFCTransaction tr = new IFCTransaction(file))
  {
    // The PlacementSetter class does the calculations to determine the local placement
    // of the element, based on the rules for the entity type.
    using (PlacementSetter setter = PlacementSetter.Create(exporterIFC, element))
    {
      // The IFCExtrusionCreationData class is used to try to create an extrusion
      // from Element BRep data, if possible.
      using (IFCExtrusionCreationData ecData = new IFCExtrusionCreationData())
      {
        ecData.SetLocalPlacement(setter.LocalPlacement);

        IFCAnyHandle prodRep = null;
        ElementId matId = ElementId.InvalidElementId;
        if (!exportParts)
        {
          // Get the category id from the element.
          ElementId catId = CategoryUtil.GetSafeCategoryId(element);

          // Get the material id from the element.  In this case, we export only one.
          matId = BodyExporter.GetBestMaterialIdFromGeometryOrParameter(
            geometryElement, exporterIFC, element);

          // This generic routine takes the element's GeometryObject and converts
          // it to an IfcProductDefinitionShape.  This does most of the work of this function.
          BodyExporterOptions bodyExporterOptions = new BodyExporterOptions(true);
          prodRep = RepresentationUtil.CreateAppropriateProductDefinitionShape(
            exporterIFC, element, catId, geometryElement, bodyExporterOptions,
            null, ecData, true);
          if (IFCAnyHandleUtil.IsNullOrHasNoValue(prodRep))
          {
            ecData.ClearOpenings();
            return;
          }
        }

        // Get the identification data for the element, and allow the user to
        // provide overrides via shared parameters.
        string instanceGUID = GUIDUtil.CreateGUID(element);
        string instanceName = NamingUtil.GetNameOverride(element,
          NamingUtil.GetIFCName(element));
```

```
            string instanceDescription = NamingUtil.GetDescriptionOverride(element,
              null);
            string instanceObjectType = NamingUtil.GetObjectTypeOverride(element,
              exporterIFC.GetFamilyName());
            string instanceTag = NamingUtil.GetTagOverride(element,
              NamingUtil.CreateIFCElementId(element));
            string footingType = GetIFCFootingType(ifcEnumType);
            footingType = IFCValidateEntry.GetValidIFCType(element, footingType);

            IFCAnyHandle footing = IFCInstanceExporter.CreateFooting(file, instanceGUID,
              exporterIFC.GetOwnerHistoryHandle(),
              instanceName, instanceDescription, instanceObjectType,
              ecData.GetLocalPlacement(), prodRep, instanceTag, footingType);
            if (exportParts)
            {
              PartExporter.ExportHostPart(exporterIFC, element, footing, productWrapper,
                setter, setter.LocalPlacement, null);
            }
            else
            {
              // Associate the material with the footing, if the material id is valid.
              if (matId != ElementId.InvalidElementId)
              {
                CategoryUtil.CreateMaterialAssociation(exporterIFC, footing, matId);
              }
            }

            // Associate the footing with the containing level.
            productWrapper.AddElement(element, footing, setter, ecData, true);

            // Create IfcOpeningElements from the extrusion data, and associate to the IfcFooting.
            OpeningUtil.CreateOpeningsIfNecessary(footing, element, ecData, null,
              exporterIFC, ecData.GetLocalPlacement(), setter, productWrapper);
          }
        }
        // We are done; commit the pseudo-transaction.
        tr.Commit();
      }
    }
```

End export

In addition to actually writing out the IFC file, the EndExport function creates any cached relations between elements determined during the element traversal. This would include, e.g., relating ducts and duct linings, relating stair components to their stair container, and placing elements in assemblies.

Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

Modifying Code

In this section, we'll look at how to convert a common IFC property set into something supported by Revit. We'll use a simple example below, although this is just one of many modifications that can be made to the code. This example is for PSet_ManufacturerTypeInformation, taken from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm, shown below:

Property Set Name	Pset_ManufacturerTypeInformation
Applicable Entities	IfcElement
Applicable Type Value	
Definition	Definition from IAI: Defines characteristics of manufactured products that may be given by the manufacturer. Note that the term 'manufactured' may also be used to refer to products that are supplied and identified by the supplier or that are assembled off site by a third party provider. This property set replaces the entity IfcManufacturerInformation from previous IFC releases.

Name	Property Type	Data Type	Definition
ArticleNumber	IfcPropertySingleValue	IfcIdentifier	Article number or reference that may be applied to a product according to a standard scheme for article number definition (e.g. UN, EAN)
ModelReference	IfcPropertySingleValue	IfcLabel	The name of the manufactured item as used by the manufacturer.
ModelLabel	IfcPropertySingleValue	IfcLabel	The model number and/or unit designator assigned by the manufacturer of the manufactured item.
Manufacturer	IfcPropertySingleValue	IfcLabel	The organization that manufactured and/or assembled the item.
ProductionYear	IfcPropertySingleValue	IfcLabel	The year of production of the manufactured item.

This translates into the function below:

```
/// <summary>
/// Initializes manufacturer type information property sets for all IfcElements.
/// </summary>
/// <param name="commonPropertySets">List to store property sets.</param>
private static void InitPropertySetManufacturerTypeInformation(IList<PropertySetDescription>
commonPropertySets)
{
  //property set Manufacturer Information
  PropertySetDescription propertySetManufacturer = new PropertySetDescription();
  propertySetManufacturer.Name = "Pset_ManufacturerTypeInformation";
  // sub type of IfcElement
  propertySetManufacturer.EntityTypes.Add(IFCEntityType.IfcElement);

  propertySetManufacturer.AddEntry(PropertySetEntry.CreateIdentifier("ArticleNumber"));
  propertySetManufacturer.AddEntry(PropertySetEntry.CreateLabel("ModelReference"));
  propertySetManufacturer.AddEntry(PropertySetEntry.CreateLabel("ModelLabel"));
  PropertySetEntry ifcPSE = PropertySetEntry.CreateLabel("Manufacturer");
  ifcPSE.RevitBuiltInParameter = BuiltInParameter.ALL_MODEL_MANUFACTURER;
  propertySetManufacturer.AddEntry(ifcPSE);
  propertySetManufacturer.AddEntry(PropertySetEntry.CreateLabel("ProductionYear"));

  if (ExportSchema == IFCVersion.IFC4)
  {
propertySetManufacturer.AddEntry(PropertySetEntry.CreateIdentifier("GlobalTradeItemNumber"));
propertySetManufacturer.AddEntry(PropertySetEntry.CreateEnumeratedValue("AssemblyPlace",
                PropertyType.Label,
typeof(Toolkit.IFC4.PsetManufacturerTypeInformation_AssemblyPlace)));
   }

   commonPropertySets.Add(propertySetManufacturer);
}
```

This function won't be called directly during the export of an element; instead it will be registered as a property set as part of:

```
private static void InitCommonPropertySets(IList<IList<PropertySetDescription>> propertySets,
IFCVersion fileVersion)
```

which in turn is called by:

```
public static void InitPropertySets(Exporter.PropertySetsToExport propertySetsToExport,
IFCVersion fileVersion)
```

The interesting things to note are:

1. PropertySetDescription contains the definition of one IFC property set. It expects to have the name of the IFC property set, and potentially a sub-element index, which will allow it to have a consistent GUID on export. Note that PropertySetDescription doesn't know about IFC schemas; the function is supposed to take the schema into account when creating the PropertySetDescription.
2. PropertySetEntry contains the definition of one IFC property. In the simplest case, the entry is defined by the name of the IFC property, which corresponds to a shared parameter in Revit of the

same name, and the CreateXXX function identifies the type of property in IFC (IfcLabel, IfcIdentifier, etc.) However, it is possible to match the IFC property to a Revit built-in parameter, or even a localized shared parameter based on the current locale of Revit.

3. InitPropertySets is the sole function called by InitializePropertySets; InitializePropertySets can be overridden by a custom exporter (as described in the next section).

Other modifications

The following is a list of other types of modifications that could be made with the exporter. This isn't intended to be an exhaustive list, or even a "top 5" list as much as a representative list of what sorts of modifications could be made.

- Support for a new IFC entity from data accessible via Revit API
- Support for elements that are non-standard IFC entities
- Support for extended properties for materials
- Better support for non-geometric data gathered by custom UI and extensible storage (e.g. file header, user information, zone information)
- Support for additional UI options to choose between different export needs

Create your own exporter

Instead of modifying the exporter directly, a user can create an exporter based on the existing IFC exporter. To do this, you will need to do the following:

1. Create a new ExporterApplication class in your custom workspace. This should look something like:

```
/// <summary>My custom exporter.</summary>
class ExporterApplication : IExternalDBApplication
{
  #region IExternalDBApplication Members
  /// <summary> The method called when Autodesk Revit exits.</summary>
  /// <param name="application">Controlled application to be shutdown.</param>
  /// <returns>Return the status of the external application.</returns>
  public ExternalDBApplicationResult
    OnShutdown(Autodesk.Revit.ApplicationServices.ControlledApplication application)
  {
    return ExternalDBApplicationResult.Succeeded;
  }

  /// <summary>The method called when Autodesk Revit starts.</summary>
  /// <param name="application">Controlled application to be loaded to Autodesk Revit
process.</param>
  /// <returns>Return the status of the external application.</returns>
  public ExternalDBApplicationResult
    OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication application)
  {
     // As an ExternalServer, the exporter cannot be registered until full application initialization. Setup
an event callback to do this
     // at the appropriate time.
     application.ApplicationInitialized += OnApplicationInitialized;
     return ExternalDBApplicationResult.Succeeded;
  }
  #endregion

  /// <summary>The action taken on application initialization.</summary>
  /// <param name="sender">The sender.</param>
  /// <param name="eventArgs">The event args.</param>
  private void OnApplicationInitialized(object sender, EventArgs eventArgs)
  {
    SingleServerService service =
      ExternalServiceRegistry.GetService(
        ExternalServices.BuiltInExternalServices.IFCExporterService) as
      SingleServerService;
    if (service != null)
    {
      IExporterIFC exporter = new MyExporter();
      service.AddServer(exporter);
      service.SetActiveServer(exporter.GetServerId());
    }
   }
}
```

2. Create a new Exporter class, to override the base exporter. This should look like:

```
public class MyExporter : Exporter
{
  public override string GetDescription()
  {
    return "My modified IFC Exporter for Revit";
  }

  public override string GetName()
  {
    return "My IFC Exporter";
  }

  public override Guid GetServerId()
  {
    return new Guid("{some GUID value}");
  }

  public override string GetVendorId()
  {
    return "MYIFCX";
  }

  public MyExporter()
  {
  }
}
```

3. Override virtual functions as necessary. These include:

```
protected override IFCFileModelOptions CreateIFCFileModelOptions(ExporterIFC exporterIFC);

protected override void InitializeElementExporters();

protected override void InitializePropertySets(IFCVersion fileVersion);

protected override bool CanExportElement(ExporterIFC exporterIFC, Autodesk.Revit.DB.Element
element);

public override void ExportElement(ExporterIFC exporterIFC, Element element);

public override void ExportElementImpl(ExporterIFC exporterIFC, Element element, ProductWrapper
productWrapper);
```