Customizing Autodesk® Revit® 2014 IFC Export Open Source Code

Angel Velez
Senior Principal Engineer, Autodesk Revit

Class summary

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.

Key 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

Note that the class hand-out contains more details; please refer to it for detailed code and instructions not in this presentation.

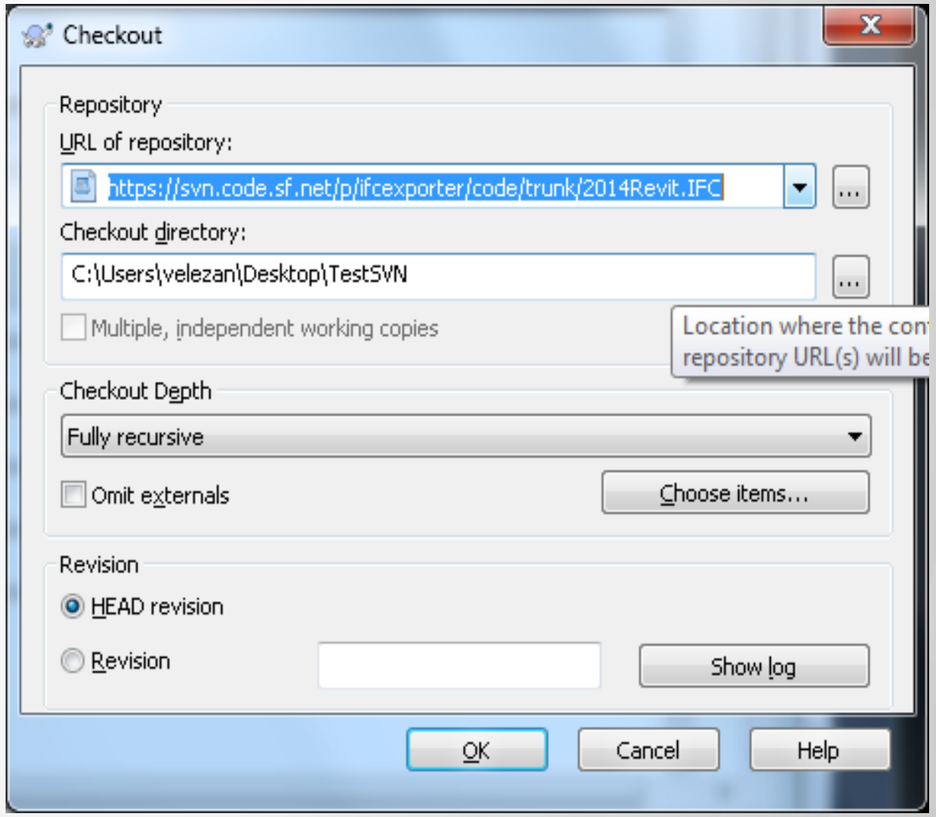
Getting the code

Where to get the code

- Two ways to get the code:
  - Download the source code ZIP file from SourceForge
    - Good if you just want to look at the code for reference
  - Get a Subversion (SVN) client
    - Recommended: TortoiseSVN from SourceForge
    - Your favorite SVN client should also work

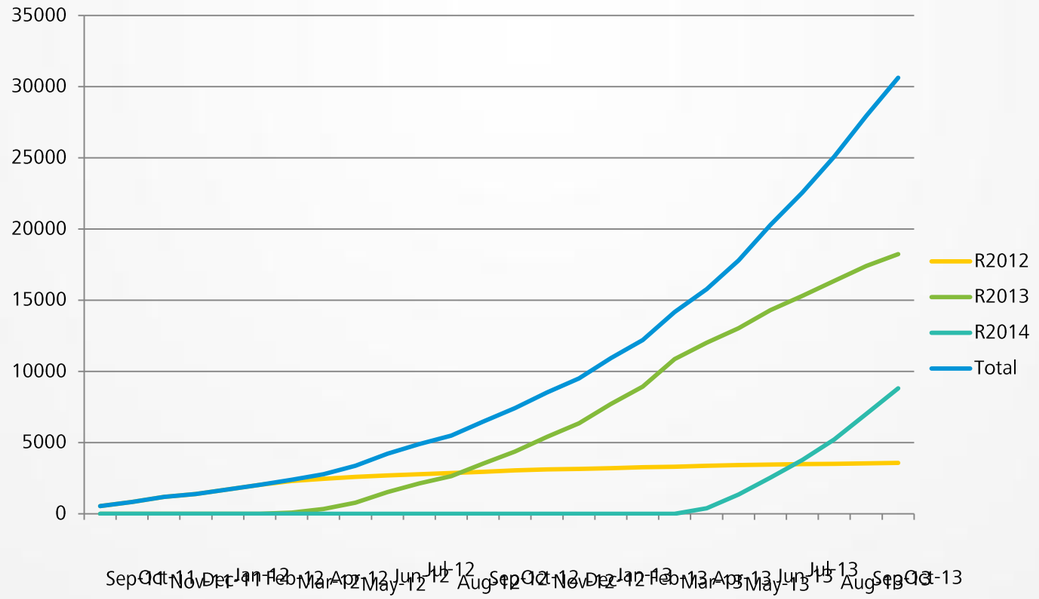
Download the code

- Create a directory for the source code
- Checkout the code
- Update the project files
  - Details in handout
- Optional: change the version number
  - Allows install to overwrite existing



Open source metrics

- No metrics for downloads via SVN of source code
- 66 downloads of v3.7.1 source code (first 30 days)
- Overall downloads from all sources: 34,000



Overall design of the Open Source code

Overview of the code

- The top level code resides entirely in Exporter.cs
- Registered as an external application via IExternalDBApplication

```
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());
  }
}
```

Overview of the code, pt. 2

- The Exporter class must inherit from IExporterIFC
  - 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:* current document being exported.
  - *exporterIFC*: initialized in native code, and allows interaction between native and .NET code
  - *filterView*: optional, for “Current View Only” export.

ExporterIFCUtils class

- IFC exporter relies entirely on Revit API
- ExporterIFCUtils extends API for IFC export-specific functionality.
  - Access to legacy elements with no API support
  - Access to what's left of the original native export
  - Over time, intend to obsolete most functions or move to other classes if generally useful

ExportIFC Function

This is the main entry point for exporting to IFC. It consists of three main parts:

- BeginExport
- Element traversal
- EndExport

```
…
BeginExport(exporterIFC, document, filterView);

InitializeElementExporters();
if (m_ElementExporter != null)
    m_ElementExporter(exporterIFC, document);

EndExport(exporterIFC, document);
…
```

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.
  - Can be overridden – we'll come back to that.
- Creating unique and top-level IFC entities, include 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 (e.g. beam systems, area schemes)
- Grids
- MEP connectors

This order can be overridden in a custom exporter.

Element traversal, pt. 2

- Each element is handled as generically as possible
- 1 or more IfcBuildingElement entities
- GUIDs are generated with the following rules:
  - 1 IfcBuildingElement = consistent IFC GUID
  - >1 IfcBuildingElement = case-by-case basis of consistent GUID
  - Code in place to ensure no GUID duplication
- Relational data may be postponed to EndExport
  - Need to have element to handle map complete before creation.

Element traversal, exporting a footing

- Footing follows general outline of exporting an element
- Handout has annotated copy of ExportFooting
- General steps:
  - Check if Parts are being exported
  - Set up top level IFCTransaction and PlacementSetter
  - Attempt to create extrusion if possible
  - Find best material for footing
  - Create IfcProductDefinitionShape from geometry
  - Create IFCFooting with GUID, name, description and other standard fields
  - Potentially create associated IfcOpeningElements

EndExport

- Creates any element relations (e.g. wall connection data) cached during element traversal
- Minor amount of internal clean-up
- Finally writes out the IFC file

Modifying the code

DYI Export, pt. 1

- 2 ways of customizing the exporter:
  - Modify the open source code directly
  - Create your own exporter that calls the open source code
- Cover method 1 in this section
- Remember to increment version number

Modify the exporter: add a property set

From http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm:

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.

Modify the exporter: add a property set, pt. 2

```
private static void InitPropertySetManufacturerTypeInformation(IList<PropertySetDescription> commonPropertySets)
{
   PropertySetDescription propertySetManufacturer = new PropertySetDescription();
   propertySetManufacturer.Name = "Pset_ManufacturerTypeInformation";
   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);
}
```

Modify the exporter: add a property set, pt. 3

- Function won't be called directly during the export of an element
- Registered as a property set as part of:

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

- PropertySetDescription contains the definition of one IFC property set
  - May have to deal with different definitions for different schemas
- PropertySetEntry contains the definition of one IFC property
- InitPropertySets is the sole function called by InitializePropertySets
  - InitializePropertySets can be overridden

Ideas for other customizations

- 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

DIY Export, pt. 2

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:

- Create a new ExporterApplication class in your custom workspace.
- Create a new Exporter class, to override the base exporter.
- Override virtual functions as necessary.

DIY Export, pt. 3

Can override a small selection of top-level functions:

- CreateIFCFileModelOptions
- InitializeElementExporters
- InitializePropertySets
- CanExportElement
- ExportElement
- ExportElementImpl