AUTODESK UNIVERSITY

AR20462

IFC Technical Overview and Survey of Autodesk Products, Including Revit 2017

Angel Velez
Autodesk, Inc.

Learning Objectives

- Explore the IFC file format, emphasizing IFC 4 and certification
- What's new in 2017 for Autodesk IFC support (including Revit)
- Understand Revit IFC options
- Learn how to make simple changes to the Revit IFC open source

Description

This class will start with a technical overview of Industry Foundation Classes (IFC) itself, with an emphasis on the new IFC4 schema and its Model View Definitions (MVDs). We will then take a high-level look at how IFC is supported across the 2017 Autodesk product line. Next we will take a look at Revit's use of IFC, and the UI options available. Finally, the class will briefly look at Revit 2017 software IFC open-source .NET code and the associated Revit software API.

Your AU Expert(s)

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. For the last 13 years, Angel has been working primarily on interoperability issues, concentrating on IFC. Since 2015, Angel has been the product owner of a Revit team responsible for core geometry and connected workflows.*

Exploring the IFC file format

Overview

IFC stands for Industry Foundation Classes. A nice summary of the history of IFC, and Autodesk's support of IFC, can be found on the Autodesk web site:

http://static-dc.autodesk.net/content/dam/autodesk/www/campaigns/interoperability/Interop_BIM%20Whitepaper.pdf

This link can be found on the general Interoperability page, which includes information about other formats such as COBie, here:

http://www.autodesk.com/campaigns/interoperability

IFC General File Format

The IFC file format is an ISO standard maintained and developed by buildingSMART. More information on the file format can be found on their web site:

IFC2x3 TC1: http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc2x3-tc1-release/summary

IFC4 Add2: http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-add2

IFC2x3 TC1 (commonly known as IFC2x3) is the most popularly supported version at the moment; IFC4 Add2 (commonly known as IFC4) is the release that supports the IFC4 Reference View and Design Transfer MVDs, and the schema which the MVDs will be certified against.

The file format itself is used more specifically for a Model View Definition (MVD). This specifies a subset of the IFC schema that is used for a particular workflow. For IFC2x3, the most common MVD is Coordination View 2.0; a technical specification of it is also on the site at:

http://www.buildingsmart-tech.org/specifications/ifc-view-definition/coordination-view-v2.0/summary

For IFC4, the two accepted MVDs are here:

IFC4 Reference View: http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-reference-view/ifc4-reference-view

IFC4 Design Transfer View: http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-design-transfer-view/ifc4-design-transfer-view

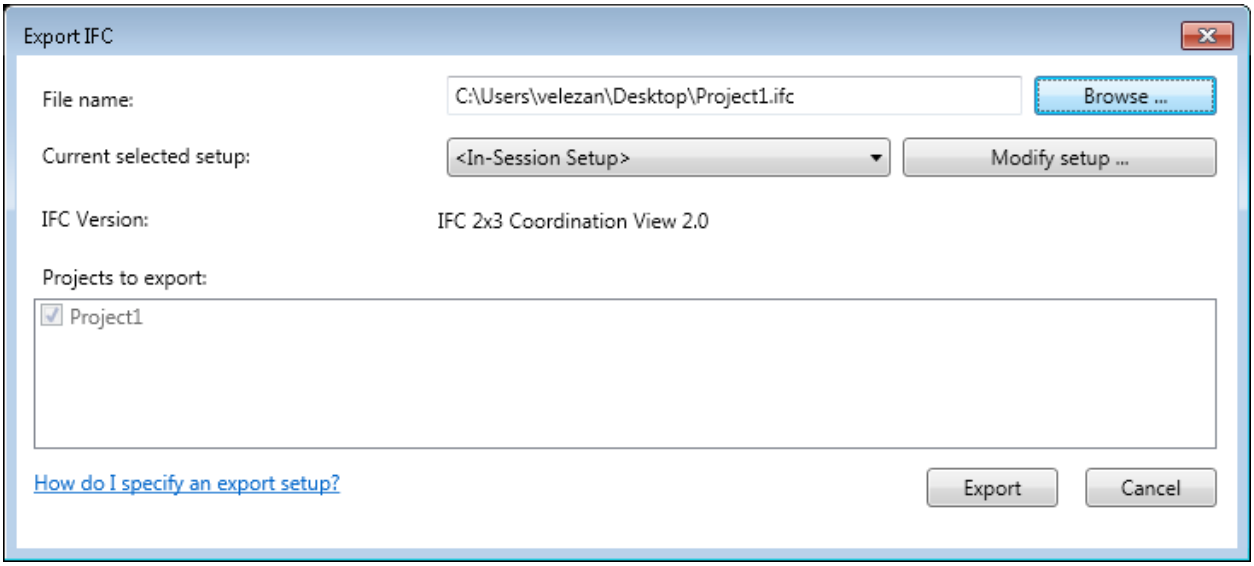
Understanding Revit IFC options

Open and Link IFC have relatively few options, which are covered in the slide deck. For Export, we cover some of the more esoteric options below. Note that the UI for export has changed considerably between Revit 2016 and 2017, and the previously "alternate" UI is now included as part of. The general functionality, though, should be similar between releases.

Export IFC

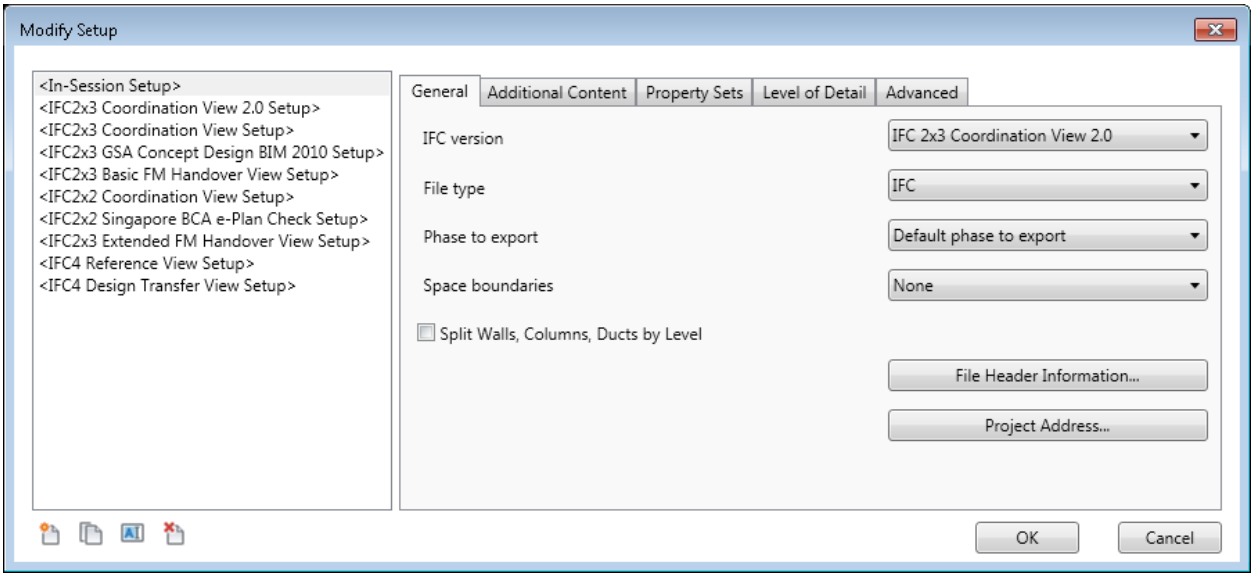
Out of the three IFC options (export, open, and link), export has the most (and the most confusing) options. Note that at the time of this writing, the Export IFC UI is undergoing significant change, and so any images associated with the export options would likely be obsolete soon. However, we intend to include updated documentation with all of the open source updates. What follows are some of the functions of some of the least obvious commands:

Main Export window:



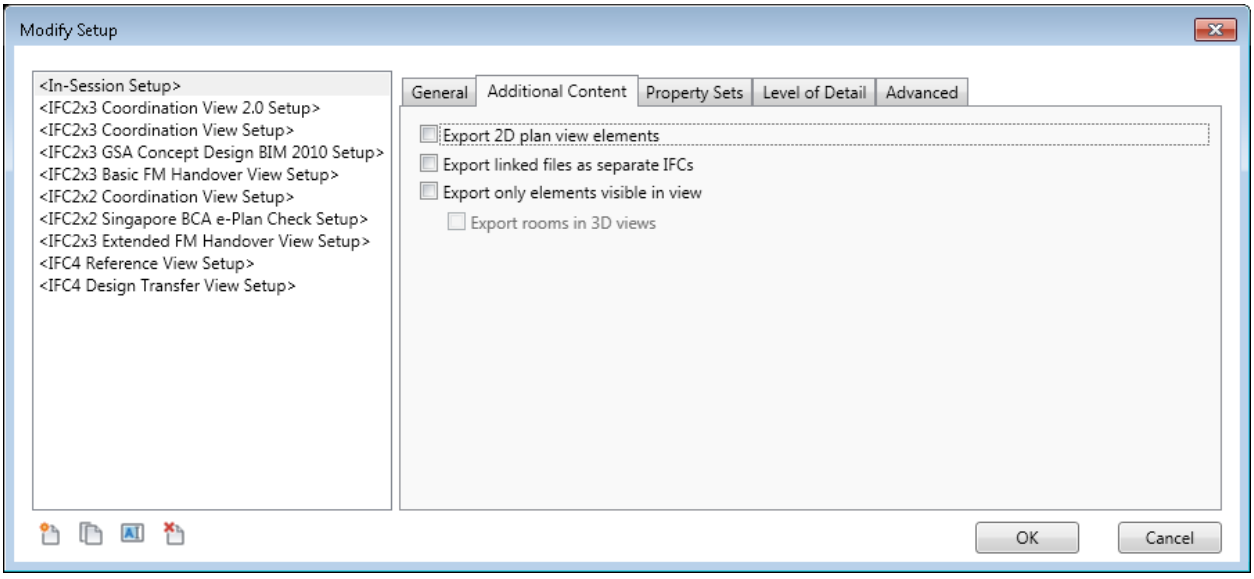
- **Projects to export**: If you do export multiple files, note that some options may not make sense (e.g., current view export) or may have flaky behavior (e.g., setting the phase to export if not all projects have that phase). Also, you won't be able to enter a file name – it will use default naming for the batch export. If you need more control, just export one file at a time, use a macro, or consider using Dynamo.

Modify Setup, General tab:



- **Project Address**: If you enter a custom address for your project when you press the ”Project Address …” button, click on “Update Project Information” if you want the information saved back to the Revit Project Information. This allows you to have the right formatting for IFC, while keeping it in sync with what’s displayed in Revit.

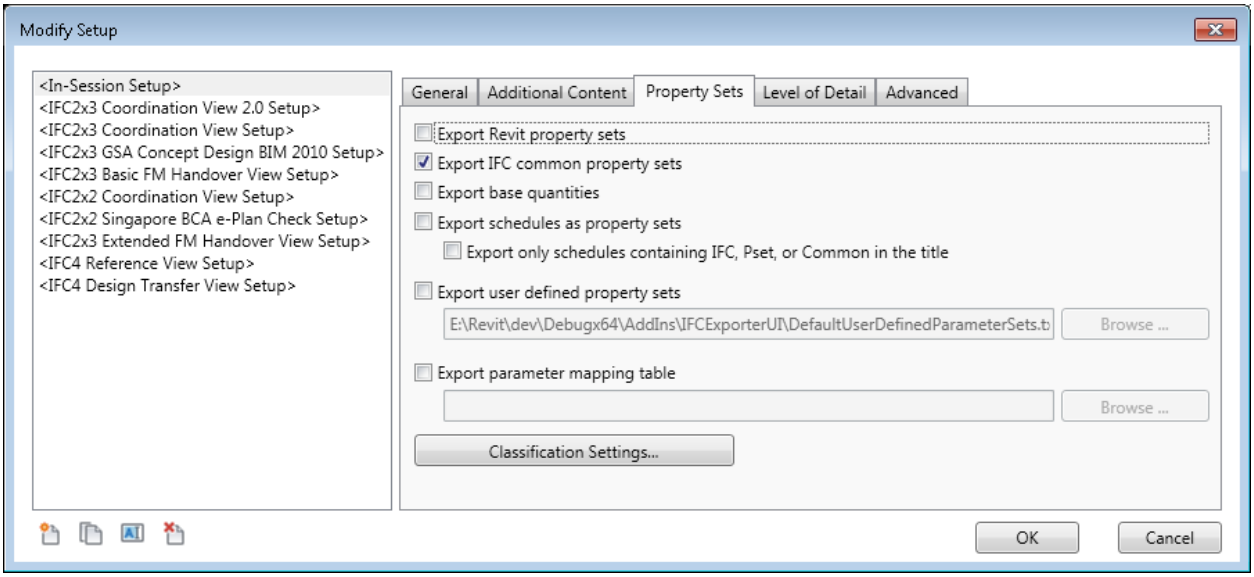
Modify Setup, Additional Content tab:



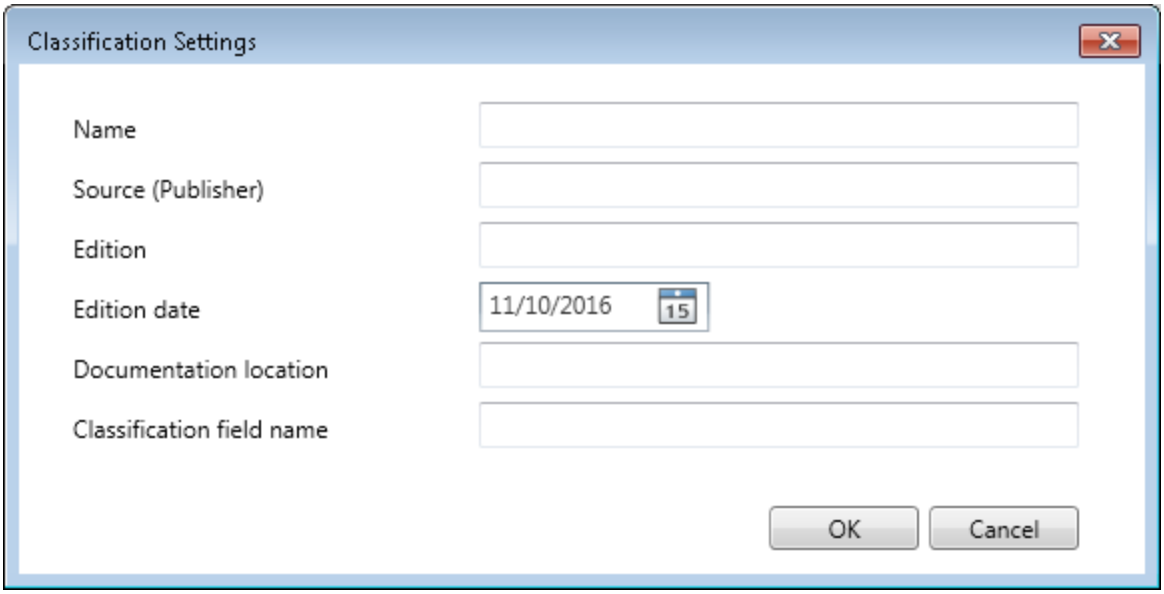
- **Export rooms in 3D views**: When the “Export only elements visible in view” option is used in a 3D view, rooms are never exported, because rooms are not visible in 3D views

in Revit. To change this behavior, check the “Export rooms in 3D views” option before export. If the 3D view has no visible section box, then all rooms will be exported. If it does have a visible section box, then only those rooms that are at least partially inside the section box are exported. Note that unlike CAD exports, IFC export always exports the entire element, so if the room is partially in the section box, the entire room is exported.

Modify Setup, Property Sets tab:

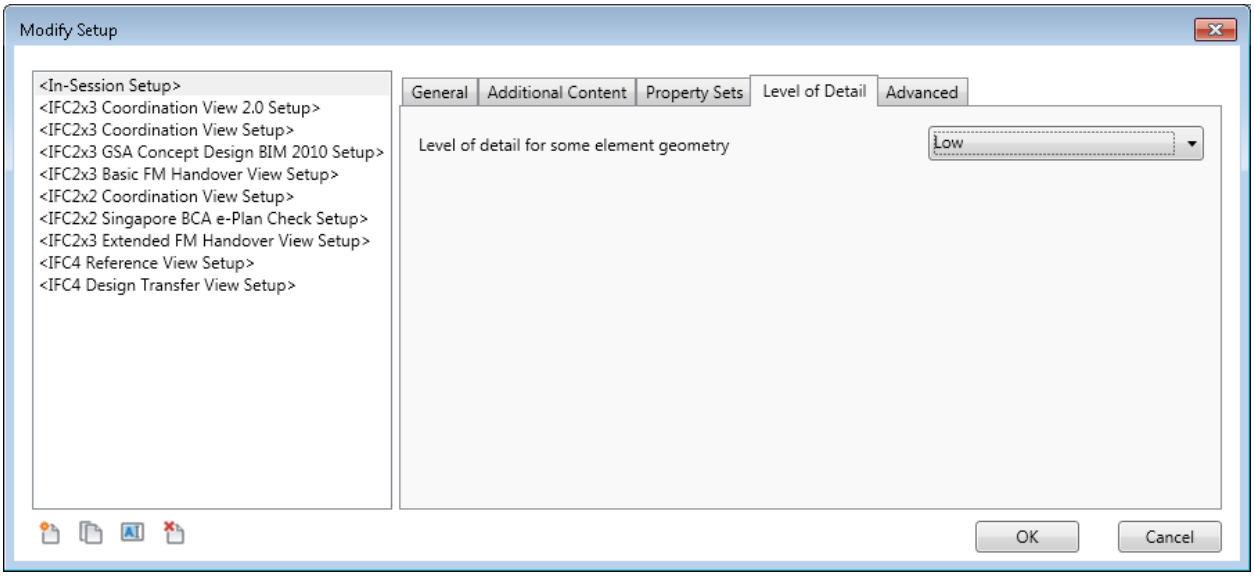


- **Export user defined property sets**: This option allows the user to specify a text file that contains the definition of one or more instance or type property sets. An example with instructions embedded as comments is included in the IFC folder when installed from either SourceForge or the AppStore, and in the slide deck associated with this handout. Note that there are some limitations to what can be defined; in particular, it doesn’t support calculated fields. By default, this option is unchecked.



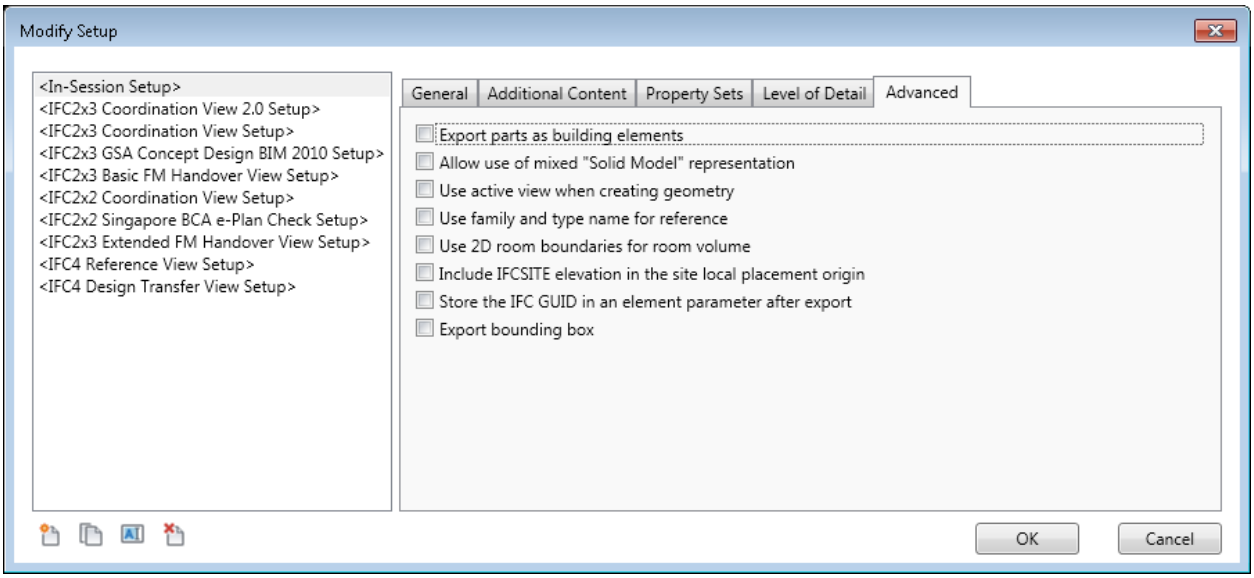
- **Classification Settings**: If you want to specify a classification Revit, you should specify the Name, Source and Edition, otherwise it isn't a valid IFC classification. In previous Revit versions, the UI would disallow you from exiting the dialog without entering all three; starting in Revit 2017, this is no longer checked. Optionally, you may enter the name of the Revit parameter that contains the classification information for the exported element(s) in the "Classification field name" field. For example, if the "Mark" parameter contains the information, enter that here. By default, it will look in the "ClassificationCode" shared parameter, if it exists. Note that for exporting Zones, Revit will look for "ZoneClassificationCode", to distinguish it from the room's classification code. The UI right now is hardwired to allow for only one custom classification field name.

Modify Setup, Level of Detail tab:



- **Level of detail for some element geometry**: This option determines two things: (1) how many facets to use when exporting faceted geometry for elements, and (2) how many points to use when approximating extrusion outlines for profiles with advanced geometry. For most cases, the “Low” default setting is a good compromise between file size and decent looking geometry. However, the “Extra Low” setting can be very useful when the file won’t export otherwise, or when speed and file size matter at the expense of appearance.

Modify Setup, Advanced tab:



- **Export parts as building elements**: The IFC2x3 schema states that elements split into Parts should be exported as a top level element (e.g., IFCWALL) that aggregates a group of IFCBUILDINGELEMENTPARTs, one per Revit Part. Some downstream applications do not properly process this information, and instead need the parts to be top-level entities (e.g., separate IFCWALLs, with no container). This option allows for this alternate representation. Only use it if you know that your receiving application can't handle IFCBUILDINGELEMENTPART – this building element parts are a generally better way of representing this information.
- **Allow use of mixed "Solid Model" representation**: When exporting an element's geometry for the IFC2x3 MVDs, the geometry may be represented by more than one extrusion, or more than one faceted BRep, but not a combination of the two. That generally means that if your Revit element has many extrusions and one item that can't be extruded, all of the extrusions must also be faceted, resulting in larger files with lower quality data (and perhaps leading to errors where the file is too big to export). Choosing this option allows the use of a "Solid Model" representation which is outside of the IFC2x3 Coordination View 2.0 MVD, but is generally supported by vendors, that allows mixing of extrusions and BReps into one geometric definition of an entity. Use this option only if you know the receiving application can support it.
- **Use active view when creating geometry**: This tells Revit to try to use the geometry of the object as seen in the current view on export. This option was created specifically for exporting certain MEP elements such as cable trays, whose model geometry differs from the displayed geometry.
- **Include IFCSITE elevation in the site local placement origin**: When exporting the site information into IFCSITE, the IFCSITE has an elevation parameter. In addition, it also has a local coordinate system. The IFC2x3 CV2.0 MVD specifies that the local coordinate system should not contain the elevation, but instead be set at Z = 0. Unfortunately, for some older systems, this resulted in the site being placed at the wrong elevation. This restores the "old" behavior of having the elevation in both places. Note that using this option in a modern system may result in the site being twice as high as it should be, so use this option with caution.

Make Simple Changes to the Revit IFC Open Source

Getting the DLLs

Autodesk has been updating the open source on a regular basis since the Revit 2012 release. README files are available on the SourceForge site with installation files and summaries of the updates to the various releases. These are found at:

2017: http://sourceforge.net/projects/ifcexporter/files/2017/

2016: http://sourceforge.net/projects/ifcexporter/files/2016/

2015: http://sourceforge.net/projects/ifcexporter/files/2015/

2014 Exporter: http://sourceforge.net/projects/ifcexporter/files/2014/

2014 Export UI: http://sourceforge.net/projects/ifcexporter/files/2014%20UI/

2013 Exporter: http://sourceforge.net/projects/ifcexporter/files/2013/

2013 Export UI: http://sourceforge.net/projects/ifcexporter/files/2013%20UI/

2012: http://sourceforge.net/projects/ifcexporter/files/2012/

Note that the 2012 version doesn't include UI, and since 2015, the exporter, export UI and importer are all in one install.

Alternatively, users can go to the Exchange Apps Store to download the installers. These are:

2017:

https://apps.autodesk.com/RVT/en/Detail/Index?id=1049118595309324136&appLang=en&os=Win64

2016:

https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3aifc2016_windows32and64%3aen

2015:

https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3aifc2015_windows32and64%3aen

2014 Exporter:
https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3aifcexporterforrevit2014%3aen

2014 Export UI:

https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3arevitifcexportalternateui2014%3aen

2013 Exporter:

https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3aifcexporterforrevit%3aen

2013 Export UI:

https://apps.exchange.autodesk.com/RVT/en/Detail/Index?id=appstore.exchange.autodesk.com%3arevitifcexportalternateui%3aen

The App Store tends to lag the SourceForge site by 1-2 weeks. Customers downloading from the App Store will get emails when new versions come out (they can choose to opt out at any time by responding to the email.) Note that the App Store only contains the compiled DLLs; SourceForge is the only place to get the source code itself.

Modifying the open source

Getting and compiling the code

There are two ways to get the source code for IFC. The best way is to download it via the source control system on SourceForge. To do this, you'll need to follow the steps below. Note that some steps are different depending on the version you are modifying, so pay close attention to the version next to some of the steps. If no version is indicated, then that is necessary for any version. Also note the 2013 and 2014 versions require Visual Studio 2010; the 2015 and 2016 versions require Visual Studio 2012, and Revit 2017 requires Visual Studio 2015.

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. **(2015+) Get the WiX Toolkit.**

Starting in 2015, the installer uses WiX. You'll need to get at least v3.7, which you can get here if you don't already have it. Keep track of the installed directory as you will need it later.

3. **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.

4. **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 line in table 1 is for the 2015 IFC code; to download the 2016 and 2017 versions, simply update the URL value with the version number . You can also modify the 2013 or 2014 exporter code, or the UI for 2013 or 2014. The directories for each are:

- **2014**: 2014 exporter code
- **2014IFCExporterUI**: Alternate UI for 2014
- **BIM.IFC**: 2013 exporter code
- **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. This is obsolete for 2014 and beyond.

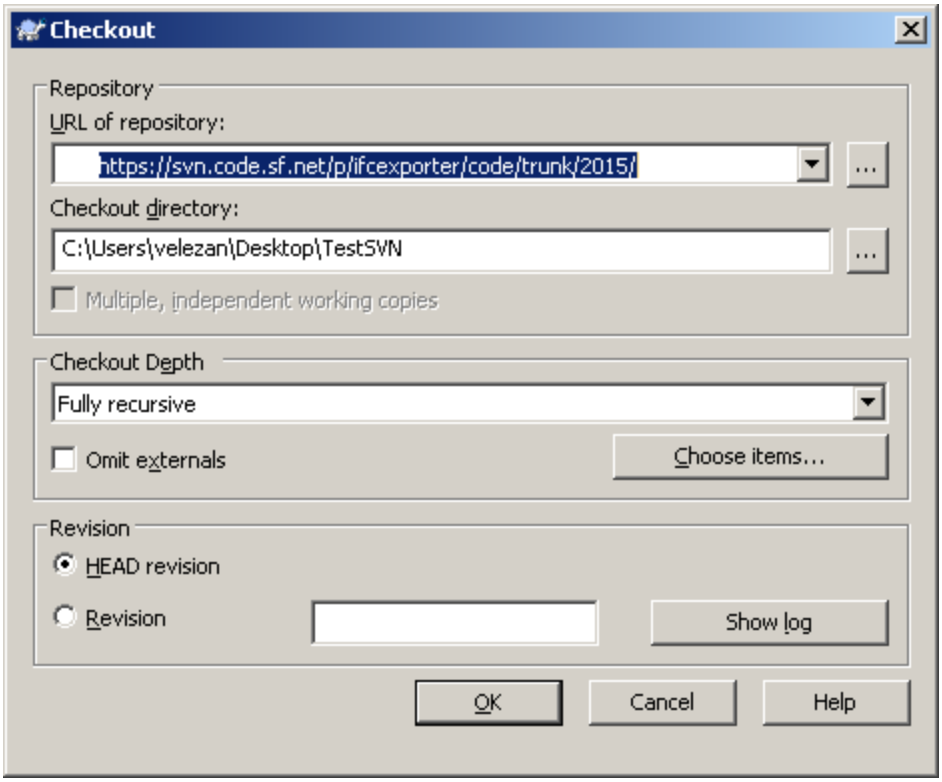


Table 1. SVN Checkout window.

5. Updating the project files – The easy way and the hard way.

5a. The easy way

We have taken the steps below and created an executable on the SourceForge site here that automates the process. Use the v2 version of the file and it should work with a few simple file paths.

5b. The hard way

Following are a few steps that are required to get the project to compile, if you prefer to make the changes manually. Note that modifying project files can be tricky; simple syntax errors may cause hard to debug issues. If you are getting hard to understand errors, a possible step is to start over. That said, the number of changes aren't that many and can be done (this method has been tested a couple of times to verify that it actually works).

5b1. Update the .props files.

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

- Source/Revit.IFC.Common/Revit.IFC.Common.props
- Source/Revit.IFC.Export/Revit.IFC.Export.props
- (2015+) Source/Revit.IFC.Import/Revit.IFC.Import.props
- (2015+) Source/IFCExporterUI/IFCExporterUI.props

- (2013/2014) Install/Modify Addin file.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. Replace the paths in those files carefully, and make sure you are using the right version of Revit.

5b2. (2015+) Update the installer files in the Install/RevitIFCSetupWix directory.

There are a few files that need to be modified in this directory. These need to be done in a text editor.

(a) buildInstaller.bat

Modify the line below:

set WixRoot=C:\Program Files (x86)\WiX Toolset v3.7\bin\

The path should match the location of your WiX library established in step 2 above. Note the extra "bin" directory where the candle.exe and light.exe executables actually live.

Next, delete the lines starting with "copy" and "del"; they are unnecessary. You can also change the line starting with "echo"" to have "Release" instead of "Releasex64", but that's optional.

(b) Product.wxs

The paths in this file needs to be modified. In particular:

(i) remove the two <Component Id …></Component> sections referring to ".sig" files.

(ii) replace "..\..\..\..\..\Releasex64" with "..\..\Source" in 5 locations (after deleting 2 above).

(c) RevitIFCSetupWix.csproj: optionally remove the 4 lines starting with "<Scc" to avoid unnecessary source control questions.

5b3. (2013 and 2014 only) Update the installer files

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.

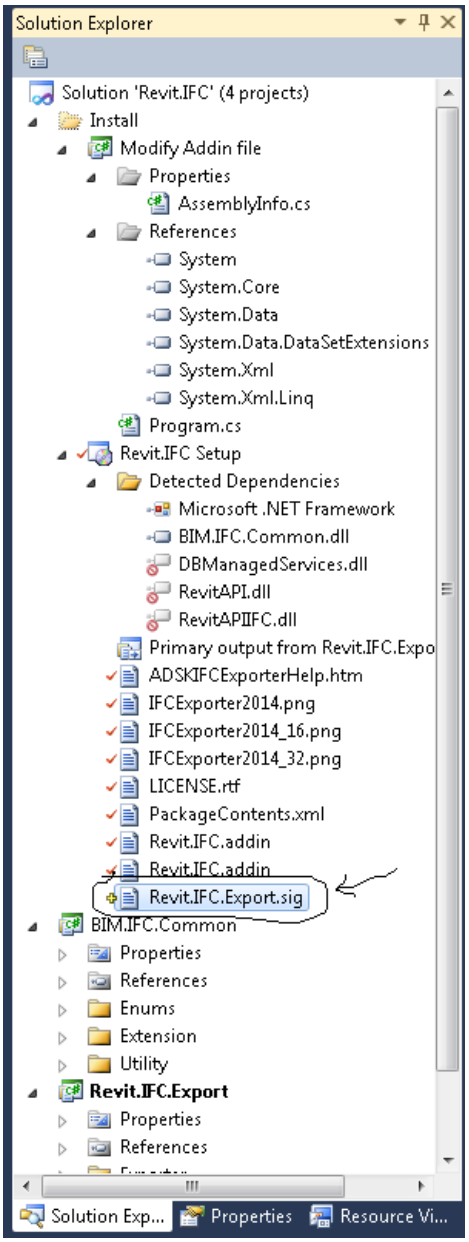


Table 2. Revit.IFC.Export.sig file.

5b4. Final clean-up for all versions

Open Revit.IFC.Export.csproj in a text editor 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. For 2015 and 2016, repeat the process for Revit.IFC.Import.csproj.

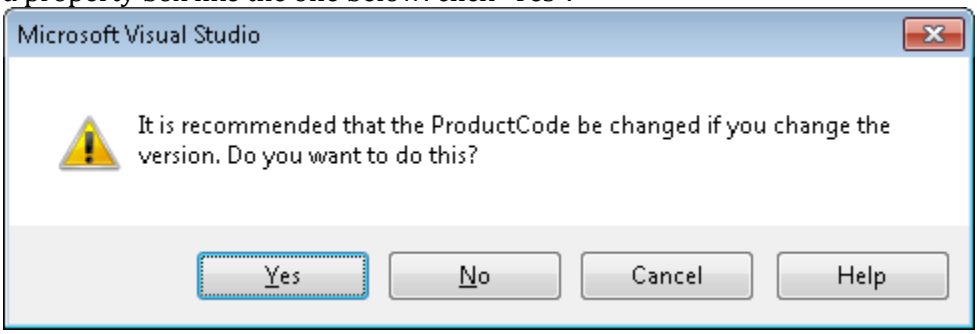
6. Build the solution

Open "Revit.IFC.Sln" in Visual Studio, and build either Debug or Release. Note that 2013/2014 use VS2010; 2015/2-16 use VS2012. This should create the installer in either Install\Setup\Debug or Install\Setup\Release, depending on your settings for 2013/2014, or in a "Debug" or "Release" folder parallel to your main folder for 2015/2016. Note that you will then need to run the created installer to install the add-in. You may get a bogus source control question here if the "Scc" sections weren't properly removed; just ignore this and disable source control.

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:

- (2015+) update the Product.wxs file with the version number. Make sure to update the ProductCode also; you can use the Create GUID tool in VS to create a new GUID.
- 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("3.12.1.0")]
  [assembly: AssemblyFileVersion("3.12.1.0")]
  ```
- (2013/2014) 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.

Revit IFC Open Source Structure

Export

The export code structure has remained relatively stable for the past couple of releases. The notes below are from the 2014 AU class DV2020: Customizing Autodesk® Revit® 2014 IFC Export 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.

```
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.

Import

The import code structure is based on two concepts: the IFC entity hierarchy and the "process/create" workflow.

IFC entity hierarchy -> Revit classes

Most of the Revit classes that represent top-level IFC entities are found in the Revit.IFC.Import/Data subdirectory. The base class for these is *IFCEntity.* This class generically represents IFC entities, parallel to the *IFCAnyHandle* class used in both import and export. For the derived classes:

1. Abstract classes in IFC are generally represented by abstract classes in Revit.
2. Not all IFC classes have been implemented in Revit, but these will be implemented as necessary.
3. There is generally a one-to-one mapping between IFC fields and Revit variables; the variable names tend to match the IFC names for simplicity.

Process/Create

The conversion of IFC data into Revit elements takes place in two passes: a Process pass, and a Create pass. This is all managed by the static IFCImportFile.Create function. The overview of the two steps follows.

All Revit classes that derive from IFCEntity have a Process function. The base function looks like this:

```
virtual protected void Process(IFCAnyHandle item)
    {
       Id = item.StepId;
       EntityType = IFCAnyHandleUtil.GetEntityType(item);
       IFCImportFile.TheFile.EntityMap.Add(Id, this);
       IFCImportFile.TheLog.AddProcessedEntity(EntityType);
    }
```

All classes are expected to override this function, and to call the base function. The IFCRoot example follows:

```
/// <summary>
/// Processes IfcRoot attributes.
/// </summary>
/// <param name="ifcRoot">The IfcRoot handle.</param>
protected override void Process(IFCAnyHandle ifcRoot)
{
  base.Process(ifcRoot);
  GlobalId = IFCImportHandleUtil.GetRequiredStringAttribute(ifcRoot, "GlobalId", false);
  if (Importer.TheCache.CreatedGUIDs.Contains(GlobalId))
    IFCImportFile.TheLog.LogWarning(Id, "Duplicate GUID: " + GlobalId, false)
  else
    Importer.TheCache.CreatedGUIDs.Add(GlobalId);
  m_Name = IFCAnyHandleUtil.GetStringAttribute(ifcRoot, "Name");
  m_Description = IFCAnyHandleUtil.GetStringAttribute(ifcRoot, "Description");

  IFCAnyHandle ownerHistoryHandle = IFCAnyHandleUtil.GetInstanceAttribute(ifcRoot, "OwnerHistory");
  m_OwnerHistory = IFCOwnerHistory.ProcessIFCOwnerHistory(ownerHistoryHandle);
}
```

The process functions are not intended to be called directly, but instead via the static ProcessIFCXXX functions that create an IFCXXX object (for example, the call to IFCOwnerHistory.ProcessIFCOwnerHistory call in the example above).

Once the entire IFC file has been processed and the IFCEntity (and derived) objects have been created, the Create process starts. In general, the Create function will create 1 Revit element per IFCEntity object (usually a DirectShape element); there is, however, no requirement that it create any element, or only 1.

Currently, the link process goes exclusively through the open source; the open path is still almost exclusively in native. However, it would be possible to overwrite the open path, or to use parametric Revit elements instead of DirectShapes for more of the elements created.

DirectShape Element

In general, the Create functions can create any type of element accessible via the API. In practice, for Link IFC, most elements created are DirectShapes. The DirectShape is an element class added in Revit 2015 which is a categorized container of externally defined geometry and parameter data.

DirectShapes generally have limited functionality compared to other native elements of the same category, and the functionality tends to improve with new releases. For example:

R2015 R2: added the ability to reference DirectShapes, especially for dimensioning. Note that not all referencing works (e.g., edge dimensions)

R2016: added the ability for DirectShapes to be room bounding. Note that unlike native walls, openings in DirectShapes affects their room bounding ability. Unlike native doors and windows, DirectShape doors and windows can be room bounding.

R2017.1: added the ability to tag DirectShapes. The leader lines, by default, go to the bounding box outline of the DirectShape, so in some cases may need to be manually modified.

As of Revit 2017.1, DirectShapes are also used when importing SAT and Rhino files.

Error Logging

An important part of the import process is accurate and timely reporting of issues. The overall goal is to have actionable items reported to the user in a non-threatening way (so, not by showing hundreds of warnings and errors in a Revit error dialog window). This logging is controlled by the IFCImportLog class, in IFCImportLog.cs. While there is no requirement to add errors in new code, it is highly recommended to allow users to ask for better models, or report issues to Revit for unsupported entities.

Revit IFC Open Source Case Studies

We'll discuss the design decisions for grids and zones in the class. The idea is not to go into all of the implementation details, but to instead discuss the philosophy behind the design. The code for these objects are in IfcGrid.cs and IfcZone.cs, respectively.

Grids

There are significant differences between IFC Grids and Revit Grids that add complexity to the conversion. The definition of IFCGrid is as follows:

> Definition from IAI: *IfcGrid* is a planar design grid defined in 3D space used as an aid in locating structural and design elements. The position of the grid (*ObjectPlacement*) is defined by a 3D coordinate system (and thereby the design grid can be used in plan section or in any position relative to the world coordinate system). The position can be relative to the object placement of other products or grids. The XY plane of the 3D coordinate system is used to place the grid axes which are 2D curves (e.g. line, circle, trimmed curve, polyline, or composite curve).
>
> The inherited attributes *Name* and *Description* can be used to define a descriptive name of the grid and to indicate the grid's purpose. A grid is defined by (normally) two or (in case of a triangular grid) three lists of grid axes.

In Revit, a Grid element represents a grid line, not an actual grid. It actually matches the IfcGridAxis entity, defined as follows:

> Definition from IAI: An individual axis, the *IfcGridAxis* is defined in the context of a design grid. The axis definition is based on a curve of dimensionality 2. The grid axis is positioned within the XY plane of the position coordinate system defined by the *IfcGrid*.

Additionally, Revit Grids:

1. Don't actually have to form a grid – they can be a single grid line. This isn't allowed in the IFCGrid definition.

2. Can't be closed circles for radial grids, which is very common for radial grids from other systems.

Most importantly, though, IFC Grids are defined per building story (IfcBuildingStorey). Revit Grids, on the other hand, are defined per project. This means that:

1. The "same" grid line (from Revit's point of view) may be defined multiple times in the file.
2. Multiple grids may overlap in Revit because there are defined on separate levels.

Zones

Revit doesn't have a generic Zone element. In this case, using DirectShapes to represent zones is ideal, since there is no alternative representation. However, given no native representation for zones (outside of HVAC), two major questions are raised:

1. How should zones be visually represented?
2. What category should zones exist in?

Zone visual representation

Zones do not generally contain their own geometry. Instead, they are a collection of spaces which individually have geometry. Ideally, then, a zone would be represented as a "container" class that maintained associativity with the contained spaces. Unfortunately, each of the Revit container classes has a limitation that makes it unusable for our purposes:

1. Assemblies have meaning on top of grouping, such as assembly views, that would make them confusing to the users who are using assembly views for their primary purpose.
2. Groups have neither assignable categories nor support for shared parameters.
3. There is no API for in-place families

To work around this, we've made the following decisions:

1. Spaces associated to zones have a parameter that contains the zone name for scheduling. This is the only actual association.
2. Zones contain the (duplicate) geometry of all of their contained items.
3. Zones are Generic Models, but all of their geometry exists in the "IfcZones" sub-category. This allows for turning off (or isolating) their geometry.