

Session 6

IFC for Interoperability

Martijn de Riet, Mdr Advies

Class Description

This class provides deeper insight of the workings of IFC in general and IFC and Revit in particular.

To be able to work with any kind of tool, one must first understand it. Fastest way to do so is reading the manual. Unfortunately, when it comes to IFC there is no manual. Not one written in a language that is understandable for people completely new to the subject.

This class will therefor start with an explanation of IFC in plain English. Where did it come from? Why was it “invented”? By whom, and for what purpose? And, most important of them all: how exactly does it work. How is an IFC model built, defined?

Once we’ve determined the way an IFC is build we can determine why Revit has such difficulties with handling IFC's. And in what way the translations between Revit and IFC can be improved.

In fact Revit has numerous tools to improve IFC handling. Most of them are well hidden and poorly documented. This class aims to provide these tools.

A few examples of these tools that we will be discussing are:

- Import/Export mapping
- Manual export override
- The use of Revit Subcategories to differentiate export to IFC Classes.
- Setting IFC layer naming with export
- Various other tweaks and settings to improve IFC handling by Revit

Besides the tools Revit provides, there are also other ways to improve IFC handling:

- Autodesk Open Source IFC Exporter
- Geometry Gym IFC Importer

This class will show that the real limitations of IFC are based on implementation issues in Revit and can in fact be solved.

During this class we will show three IFC enhancements:

1. Export mapping of custom Revit parameters to standard IFC Properties.
2. The ability to create custom Property Sets and map Revit parameters to them.
3. The import of an IFC file while keeping parametric constraints.

The last subject this class will touch upon is troubleshooting. It's no secret that Revit and IFC are not a match made in heaven. Even with the handles provided in this class, things will often go wrong. So what do you do when things go horribly wrong? Are there general rules to follow when working with IFC?

About the Speaker:

Martijn de Riet is a self-employed BIM Consultant from the Netherlands, working with Revit since version 5.1.

Martijn has a bachelor degree in Building Science. After his study he started his own engineering firm working for contractors, architects and private clients.

Starting 2007 his company transformed into a full-time BIM consultancy service. At the moment Martijn's clients vary from mid-sized architectural firms to the largest Dutch General Contractor and MEP Engineering firms, with a focus on specific corporate solutions, design and implementations of Revit and BIM workflows.

Martijn is a member of the Dutch Revit User Group, co-author of the Dutch RUG "Best Practises" for Revit and author / creator of the Dutch RUG Revit Standards Master Template.

He provides lectures for companies, technical universities, seminars and such on a regular basis and will be speaking about IFC at the upcoming Autodesk University.

What is IFC?

Short and simple? IFC is a database format. Just as the .rvt format is for Revit. Or the .pln format for Archicad. Or the gbXML format for that matter.

It's just a way to store and retrieve data. And in this particular case, building data.

Now let's look at the long and complicated way to explain this. In the next image, the official IFC scheme is shown (also to be found at: <http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm>, which is probably better for your eyes).

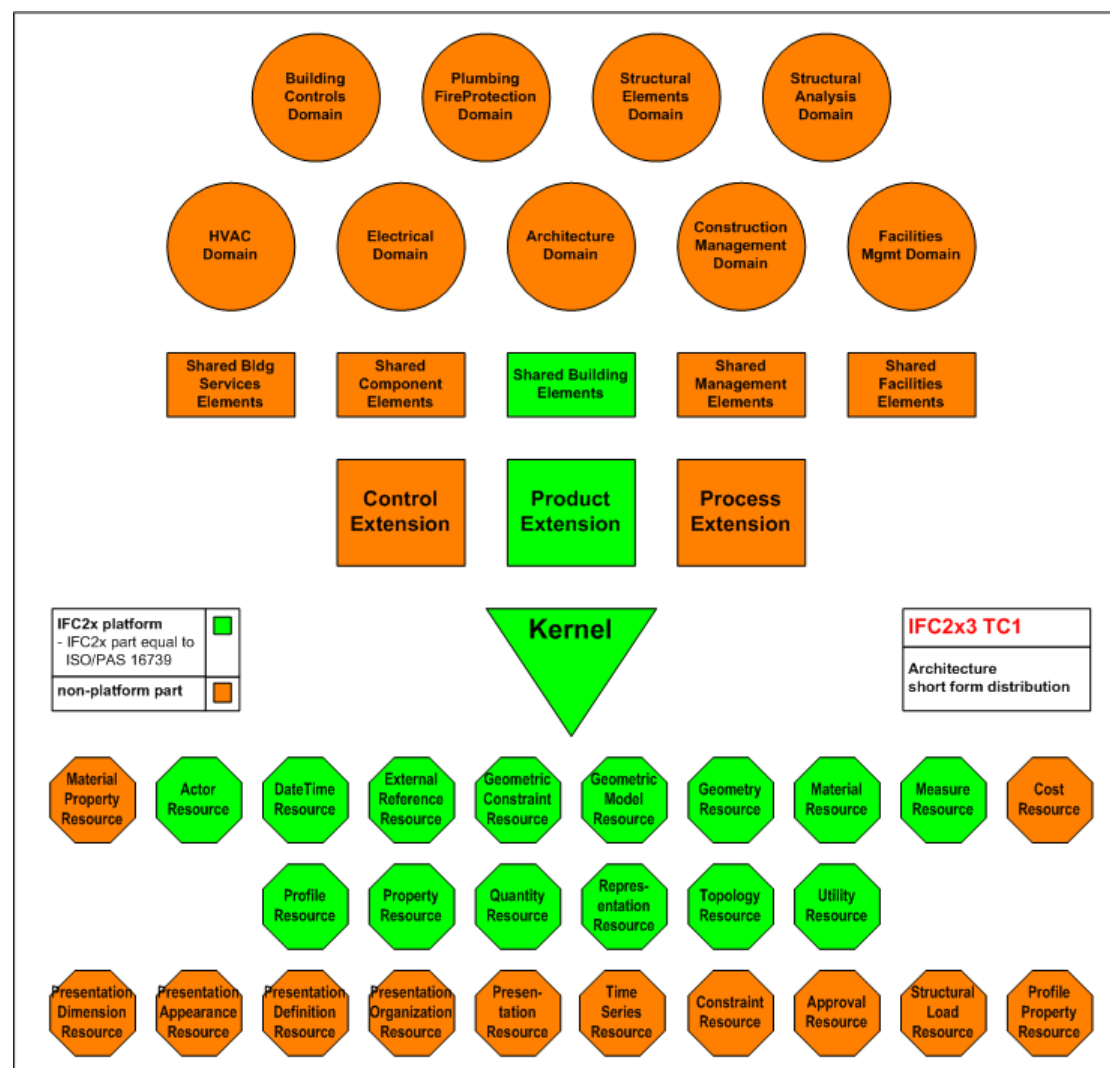


Image 1 Basic principles of IFC

Now let's try to explain this.

Kernel

Most important part of the IFC format is the "Kernel". This is the IFC Core, which basically defines all major components. Not the actual building components, but it defines the master structure of an IFC:

- An element in IFC is an Actor, Control, Process, Product or a Project.
- An element in IFC has relationships with other elements. Definitions for those relationships are found in the Kernel.
- An element has dynamic properties defined in the Property Sets. The concept of Property Sets is also part of the Kernel.

At this level of abstraction, the Kernel is really not something useful to you as a user. You might page through it, but it would probably cost you a severe headache with limited results. However, it is the part that makes IFC "smart".

Resources

Below the Kernel are the Resources. Resources are just that: possible data items that can be part of an object in IFC. There's some that are quite obvious such as *IfcCalenderDate*. Others are a complete riddle to me. To be honest, most of them are actually...

Important thing to remember: this is **not** stuff that you want to manually toy with.

Extensions

Above the Kernel you will find the Extensions. Extensions define the highest level of components for Products, Processes or Controls.

Some of these you will use (such as *IfcSite* or *IfcSpace*). But most are broken down into more specified Classes in the levels above: the Elements and the Domains.

Elements

Elements are somewhat of a go-in-between. Not Extensions (abstract level), and not Domains (highly specified). Extensions are the smaller branches in a tree. For the larger, more obvious building components they are the end of the line (Shared Building Elements, which contains Columns, Roofs, Walls, Doors, Windows and such). For parts of a building that have a more subtle definition, Extensions form a collective level. For instance:

In the Shared Building Services Extension you will find the `IfcFlowTerminal`, which in turn is the supertype for the `IfcAirTerminal` and `IfcGasTerminal` that both reside in the HVAC Domain.

Domain

These are the most important part of the IFC definition for you as a user. These define the major “editable” parts of IFC: the Classes (e.g. `IfcWall`).

How does it all work together

When you take the time to study the Architectural diagram as shown in Image 1, you will notice that IFC has a hierarchical tree-like structure.

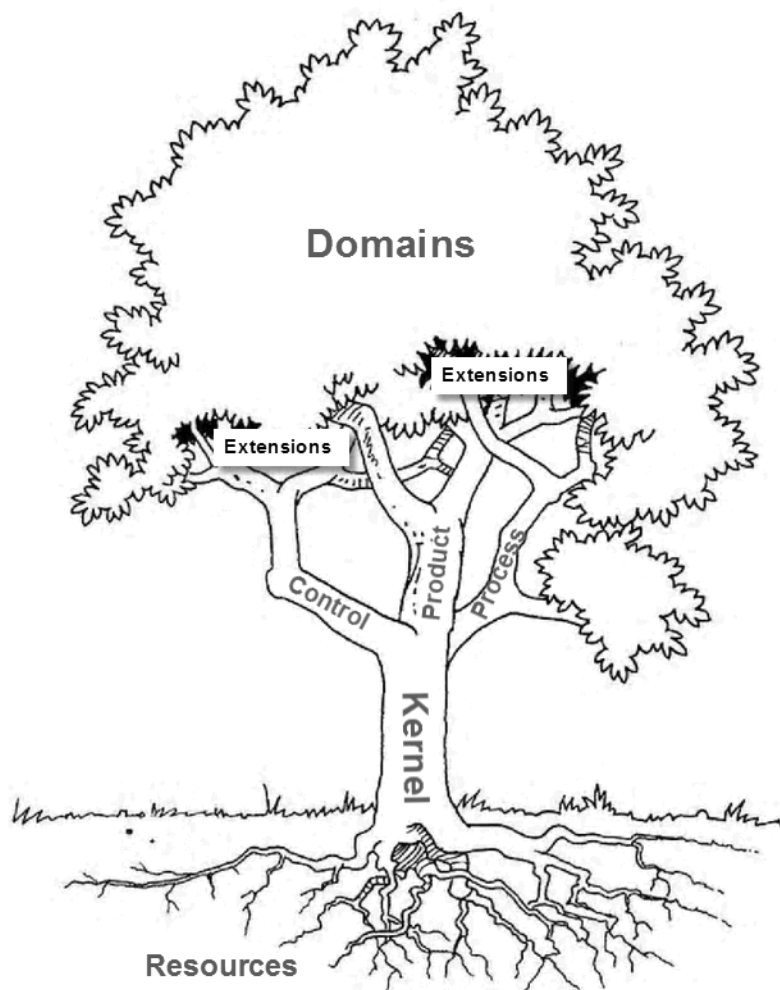


Image 2 Basic principles in a less schematic manner

If the Kernel is considered to be the tree trunk, the resources are its roots. These are the elements that feed the trunk and branches with information.

The tree trunk contains the highest level of definitions, e.g. `IfcProject` and defines the way branches are connected to each other.

The major support branches are defined in the Extensions. These contain some dead ends (`IfcSpace`) that are not further refined in the Domains. But most are. For instance the `IfcBuildingElement` which is the supertype for all major Building Components.

The Origin of IFC

Acronym

So where did this all come from? IFC is the acronym for Industry Foundation Class. Frankly, I have no idea what this means. But it has to mean something, so let's try this:

Industry will probably mean the AEC industry.

Foundation could be a non-profit organisation (not likely), or could mean that this is the fundamental language to describe a building.

Class could be short for classification.

This would add up to IFC being a common language for the AEC industry to describe a building in a database. That does sound somewhat right to me.

History

The IFC-initiative was started by the IAI (International Alliance for Interoperability). The driving force behind the establishment of the IAI in 1995 was, believe it or not, Autodesk. They needed a common language to facilitate the data exchange between different software packages (in their portfolio).

The IFC database structure is derived from the STEP-format which is described in ISO 10303. The STEP format is recognised to be the worldwide standard on "computer-interpretable representation and exchange of product manufacturing information". In common English: STEP is a vendor-independent language to describe 3D geometry and associated data needed to *manufacture* it (which is a broader description than *build*).

Intended use

The IFC format was originally intended to hand over data to **downstream** applications. In other words, IFC is specifically meant for data exchange between different project phases, from design to construction to Facility Management. That's why, if you look at Image 1, a huge part of the IFC structure is reserved for processes during construction and facility management.

So, here's the first bombshell:

IFC is **not** meant to act as an round-tripping format between project partners on the same "design level".

Now before you (virtually) tear this handout to shreds, please allow me to make an argument:

IFC is based on the assumption that the exchange model needs to contain **objects**. You simply cannot exchange sole properties. They need to be tied to a geometric shape.

This means that if you exchange IFC's you will by definition exchange objects. Whilst most of the time, what you want to exchange are object **definitions**. Take the following flow chart as a simple example.

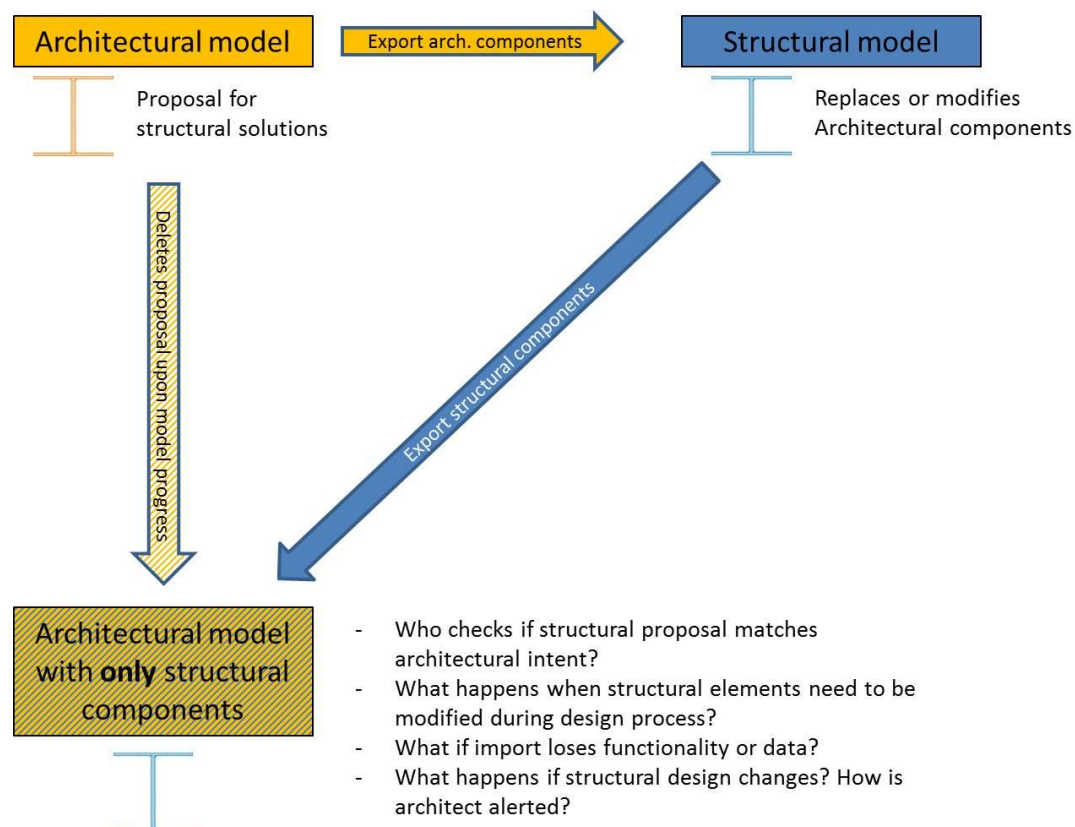


Image 3 Initial exchange between architect and structural engineer

Looks kind of right? It does have some flaws, mainly regarding model coordination. Basically this means the architect manually coordinates correct placement of structural elements. Which can be handled through clash detection or accepting duplicates, as shown in the next image.

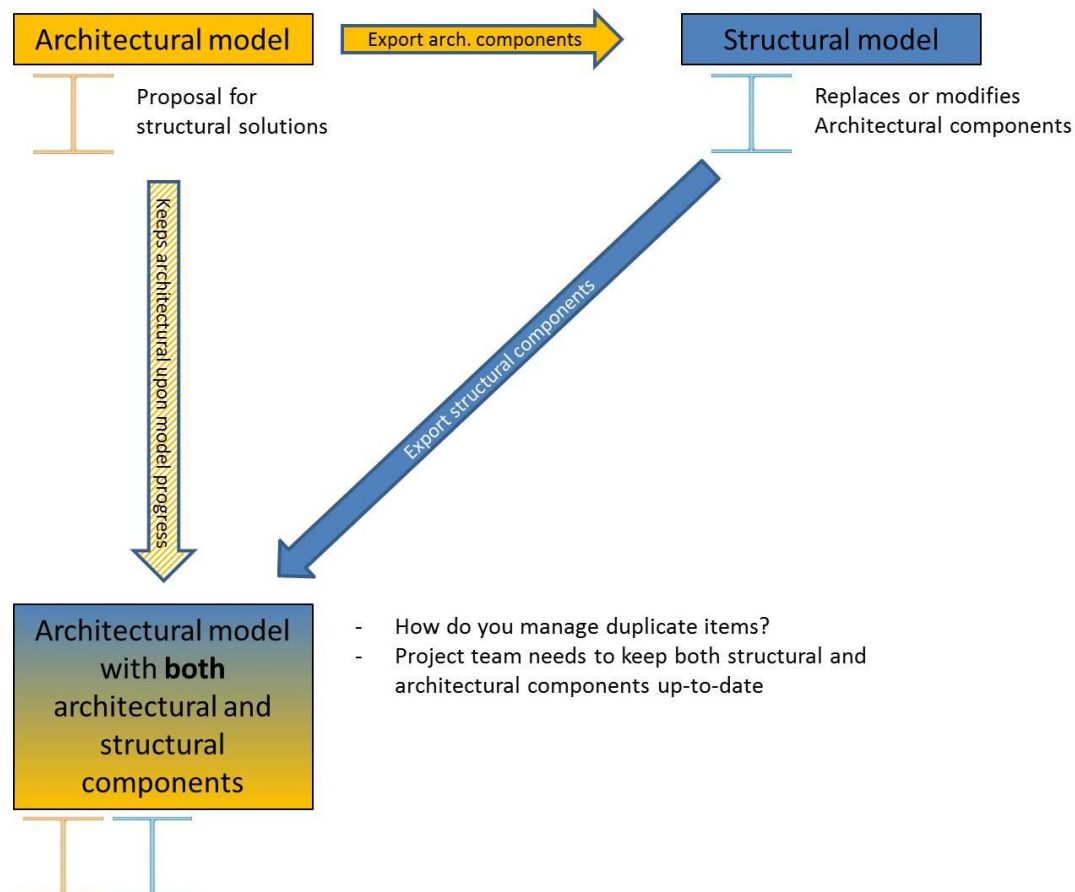


Image 4 Same process, while accepting duplicate structural elements in arch. model

This makes coordination during design a lot easier. The architect can keep their structural elements intact and just reference the structural model to see if everything is in place. However, the structural engineer is still going to need to toss his model and start over whenever there is a design change. OR, we use option 3, as shown in the next image.

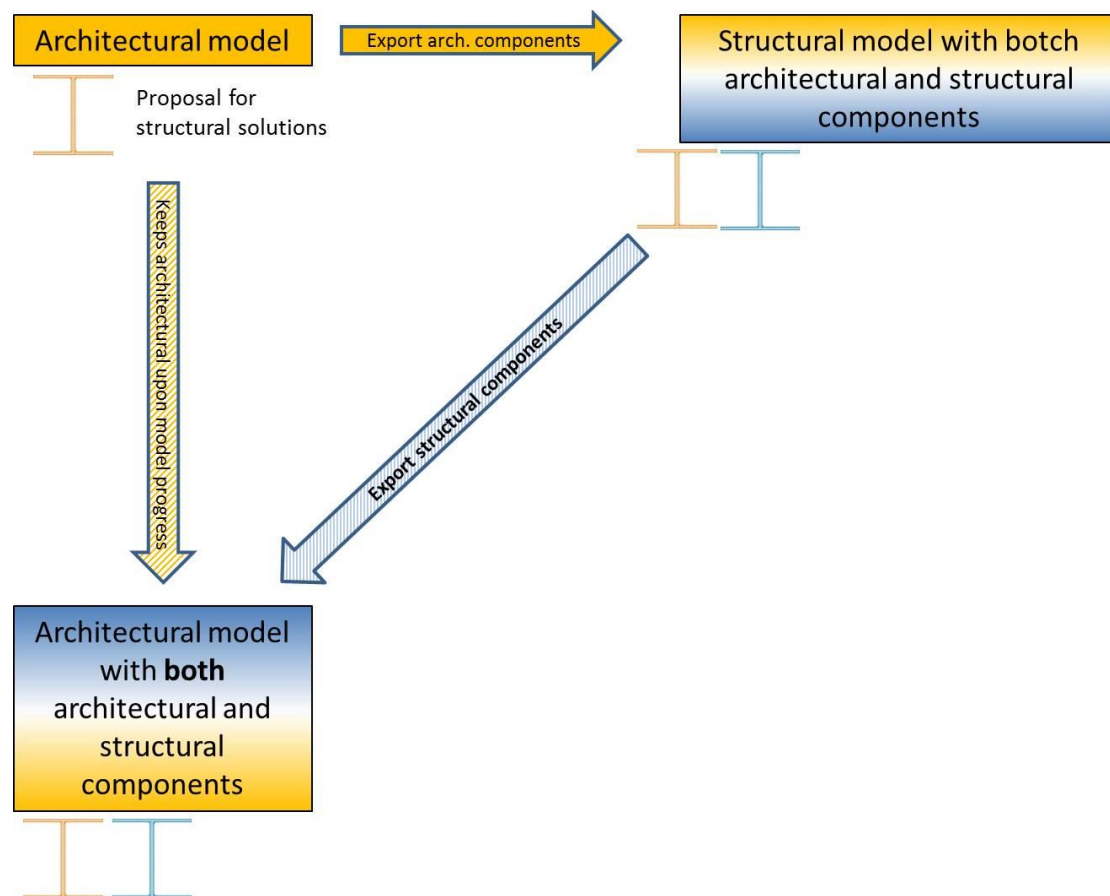


Image 5 both parties use partner model for reference

Cool. We sorted it. Both partners can develop their model without having to fear to start over when design changes. And you have your partners' model for reference.

But wait a sec. Wasn't this whole BIM-thing meant to integrate workflows? This doesn't seem very integrated to me or does it? Are we introducing a cure that's even worse than the disease?

But the point is: because IFC assumes you want to exchange geometry and not just update properties, it's not suitable for regular updates. However, that can be fixed. Just keep reading...

How does it work?

Note: if you're not interested in the technicalities behind it all, skip this part. We're going to look at the inner workings of IFC. So put your pillow in place, this might get somewhat dull and geeky.

In the dataset you will find a few IFC files. Open up *ifc exporter test 2.ifc* in Notepad (right click > open with) and an IFC viewer (I use Solibri Viewer). You will see the following picture for Solibri.

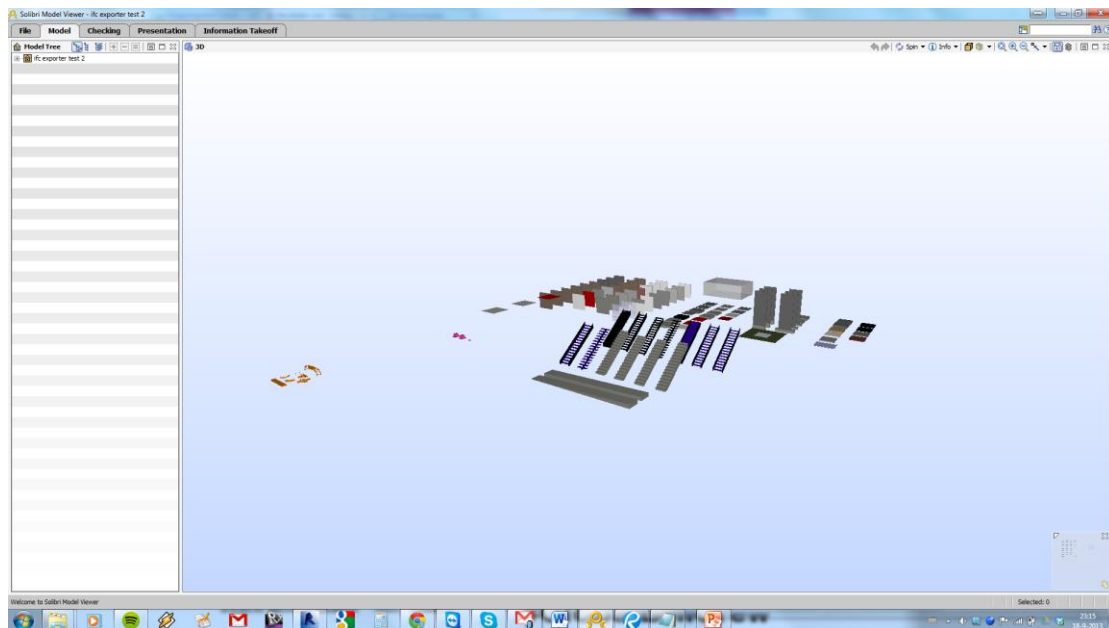
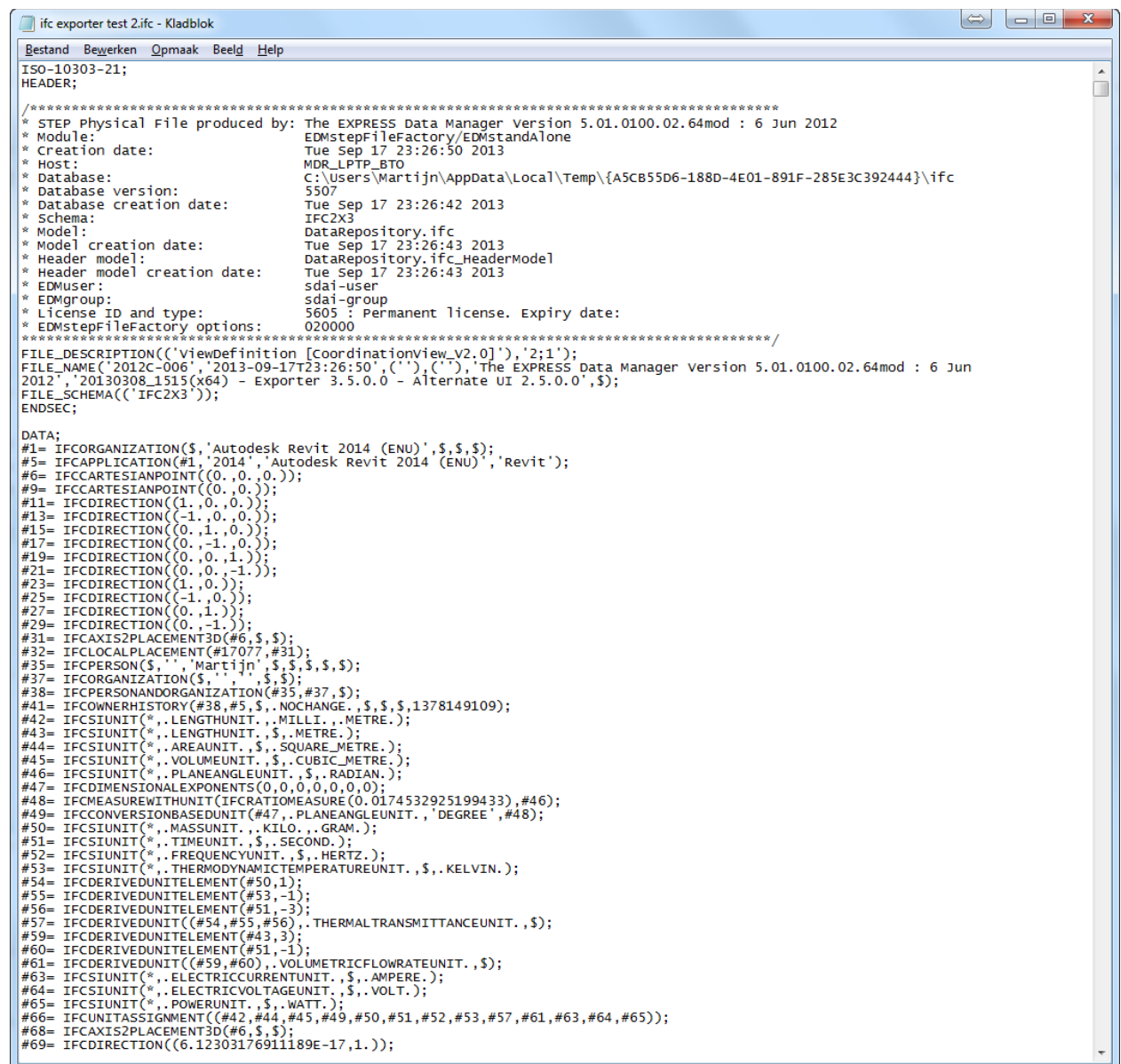


Image 6 *ifc exporter test 2.ifc* in Solibri

Looks normal right? It's the Legend View of the current version of the Dutch Revit Standards template file. It basically has some System Families and a few MEP component families, nothing fancy.

Now check the Notepad.



```

ifc exporter test 2.ifc - Kladblok
Bestand Bewerken Opmaak Beeld Help
ISO-10303-21;
HEADER;
/*****
* STEP Physical File produced by: The EXPRESS Data Manager Version 5.01.0100.02.64mod : 6 Jun 2012
* Module: EDMstepFileFactory/EDMstandAlone
* Creation date: Tue Sep 17 23:26:50 2013
* Host: MDR_LPTP_BTO
* Database: C:\Users\Martijn\AppData\Local\Temp\{A5CB55D6-188D-4E01-891F-285E3C392444}\ifc
* Database version: 5507
* Database creation date: Tue Sep 17 23:26:42 2013
* Schema: IFC2X3
* Model: DataRepository.ifc
* Model creation date: Tue Sep 17 23:26:43 2013
* Header model: DataRepository.ifc_HeaderModel
* Header model creation date: Tue Sep 17 23:26:43 2013
* EDMuser: sdai-user
* EDMgroup: sdai-group
* License ID and type: 5605 : Permanent license. Expiry date:
* EDMstepFileFactory options: 020000
*****/
FILE_DESCRIPTION(('ViewDefinition [CoordinationView_V2.0]'),'2;1');
FILE_NAME('2012c-006','2013-09-17T23:26:50','(',')','The EXPRESS Data Manager Version 5.01.0100.02.64mod : 6 Jun
2012','20130308_1515(x64) - Exporter 3.5.0.0 - Alternate UI 2.5.0.0',$);
FILE_SCHEMA(('IFC2X3'));
ENDSEC;

DATA;
#1= IFCORGANIZATION('$','Autodesk Revit 2014 (ENU)',,$,$);
#5= IFCAPPLICATION(#1,'2014','Autodesk Revit 2014 (ENU)','Revit');
#6= IFCARTESIANPOINT((0.,0.,0.));
#9= IFCARTESIANPOINT((0.,0.));
#11= IFCDIRECTION((1.,0.,0.));
#13= IFCDIRECTION((-1.,0.,0.));
#15= IFCDIRECTION((0.,1.,0.));
#17= IFCDIRECTION((0.,-1.,0.));
#19= IFCDIRECTION((0.,0.,1.));
#21= IFCDIRECTION((0.,0.,-1.));
#23= IFCDIRECTION((1.,0.));
#25= IFCDIRECTION((-1.,0.));
#27= IFCDIRECTION((0.,1.));
#29= IFCDIRECTION((0.,-1.));
#31= IFCAXIS2PLACEMENT3D(#6,$,$);
#32= IFCLOCALPLACEMENT(#17077,#31);
#35= IFCPERSON($,'','Martijn',,$,$,$,$);
#37= IFCORGANIZATION($,'','',$,$);
#38= IFCPERSONANDORGANIZATION(#35,#37,$);
#41= IFCOWNERHISTORY(#38,#5,$,NOCHANGE,,$,$,1378149109);
#42= IFCSIUNIT(*,LENGTHUNIT,,$,MILLI,,$,METRE.);
#43= IFCSIUNIT(*,LENGTHUNIT,,$,METRE.);
#44= IFCSIUNIT(*,AREAUNIT,,$,SQUARE_METRE.);
#45= IFCSIUNIT(*,VOLUMEUNIT,,$,CUBIC_METRE.);
#46= IFCSIUNIT(*,PLANEANGLEUNIT,,$,RADIAN.);
#47= IFCDIMENSIONALEXONENTS(0,0,0,0,0,0,0);
#48= IFCMEASUREWITHUNIT(IFCRATIOMEASURE(0.0174532925199433),#46);
#49= IFCCONVERSIONBASEDUNIT(#47,PLANEANGLEUNIT,DEGREE,#48);
#50= IFCSIUNIT(*,MASSUNIT,,$,KILO,,$,GRAM.);
#51= IFCSIUNIT(*,TIMEUNIT,,$,SECOND.);
#52= IFCSIUNIT(*,FREQUENCYUNIT,,$,HERTZ.);
#53= IFCSIUNIT(*,THERMODYNAMICTEMPERATUREUNIT,,$,KELVIN.);
#54= IFCDERIVEDUNITELEMENT(#50,1);
#55= IFCDERIVEDUNITELEMENT(#53,-1);
#56= IFCDERIVEDUNITELEMENT(#51,-3);
#57= IFCDERIVEDUNIT((#54,#55,#56),THERMALTRANSMITTANCEUNIT,,$);
#59= IFCDERIVEDUNITELEMENT(#43,3);
#60= IFCDERIVEDUNITELEMENT(#51,-1);
#61= IFCDERIVEDUNIT((#59,#60),VOLUMETRICFLOWRATEUNIT,,$);
#63= IFCSIUNIT(*,ELECTRICCURRENTUNIT,,$,AMPERE.);
#64= IFCSIUNIT(*,ELECTRICVOLTAGEUNIT,,$,VOLT.);
#65= IFCSIUNIT(*,POWERUNIT,,$,WATT.);
#66= IFCUNITASSIGNMENT((#42,#44,#45,#49,#50,#51,#52,#53,#57,#61,#63,#64,#65));
#68= IFCAXIS2PLACEMENT3D(#6,$,$);
#69= IFCDIRECTION((6.12303176911189E-17,1.));

```

Image 7 ifc exporter test 2 in Notepad

Ok, this might give you a headache. But no worries, I'm not about to explain everything, just a few basics.

On the top you'll find the file header with all project specs. Then, up until line #100, there are some basic settings: units, project base point, and such. Nothing to worry about.

Now let's try if we can find a component. For the purpose of this exercise I did some spinning and zooming and selected a Wall. Double clicking it will bring up the properties screen.

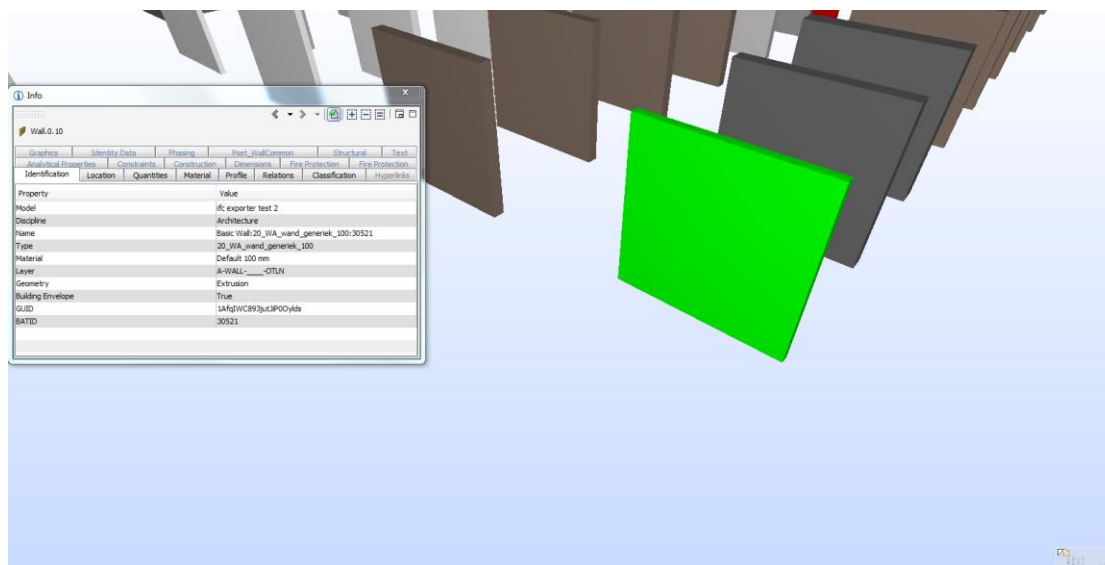


Image 8 Selecting a Wall brings up the properties

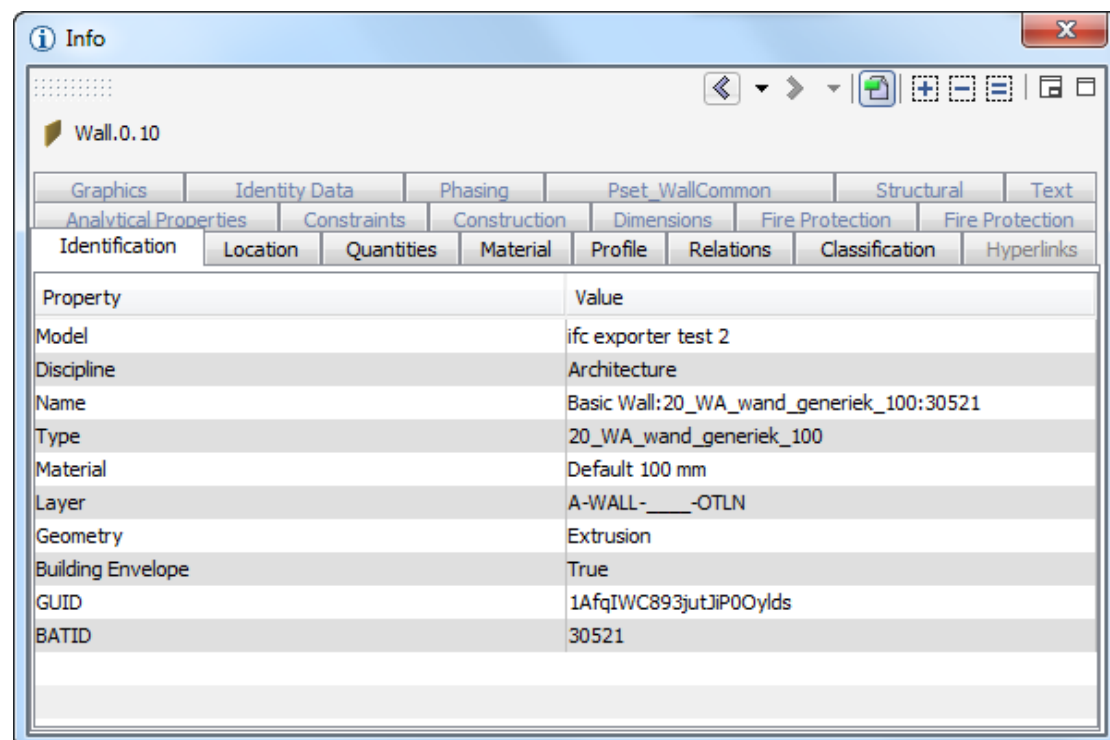


Image 9 and just the properties

This Wall has a Name, which ends with an Identifier "30521". Switch to Notepad and use Ctrl+F to find that particular Wall. You will find it at line #170. Check this by referencing the underlined text with the Name and Guid as shown in the above image.

```
#170= IFCWALLSTANDARDCASE('1AfqlWC893jutJiP0Oylds', #41, 'Basic  
Wall:20_WA_wand_generiek_100:30521', $, 'Basic  
Wall:20_WA_wand_generiek_100:1384', #139, #166, '30521');
```

What does this line of code tell us? A few things:

1. It references line #41:

```
#41= IFCOWNERHISTORY(#38, #5, $, .NOCHANGE., $, $, $, 1378149109);
```

Which I'm guessing means that there hasn't been any update (intriguing piece of information, is it not?)

2. It references lines #139 to #166. I won't bother to copy-paste them here, but these lines of code basically give the placement coordinates and define the way the solid representing this Wall is constructed.

3. The code for this Wall goes on with defining its properties and the property sets they belong in from line #170 onwards. It ends with line #298:

```
#298= IFCRELDEFINESBYPROPERTIES('3yFzT6fBf5yBXemS0KPVgE', #41, $, $, (#170), #296);
```

This tells us that the properties defining this Wall can be found between lines #170 en #296. After that, line #301 starts defining a new Wall.

Cool, so what?

Assuming that an IFC is generated by Solibri in numerical order this is the order in which objects are created:

1. Object representation and placement
2. Properties belonging to the object.

In other words: you cannot simply export the properties, you need to define the object first.

However, there is hope. Open up *ifc exporter test 3.ifc*. Find Wall number 30521 as shown:

```
#170= IFCWALLSTANDARDCASE('1AfqlWC893jutJiP0Oylds', #41, 'Basic  
Wall:20_WA_wand_generiek_100:30521', $, 'Basic  
Wall:20_WA_wand_generiek_100:1384', #139, #166, '30521');
```

Notice that the Guid and Identifier are similar even though this is a separate export. This means that it is theoretically possible to match all objects in an IFC and compare properties. And update them...

Arguing

There's this big, ongoing fight about whether we should, or shouldn't, use IFC in our projects. Before going into the practical use of this class, let's look at some of the most used arguments.

Why NOT use IFC?

1. Doesn't everybody use Revit? So why bother?
2. Loss of data.
3. Loss of intelligence.
4. In case nr. 1 doesn't fly, why not use a bi-directional link?

Why use IFC?

1. Doesn't every one use Revit?

Ehm, no. Not everybody uses Revit. Nor will they ever. For several reasons:

- Freedom of choice (user preference).
- Simple math:
Autodesk has 7000 employees. How many of them would actually be in Revit product development?
There are approximately 6.999.993.000 people not working for Autodesk. Fact of the matter is, it's far more likely that someone **not** working for Autodesk has a brilliant idea and develops your next favourite piece of software. And that it's not Revit. That's called progress. Or diversity.
- It's not the best for everything.
I love Revit. Really do. But seriously? Revit gets it's ass kicked by specialised software on all kinds of subjects:
 - Tekla is (still) better when it comes to shop drawings for structural rebar and/or steel structures. And with general modelling/construction design, it's at the very least equally capable.
 - Rhino/Grasshopper can create geometric shapes that will take forever to recreate in Revit, assuming you will eventually get there.
 - DDS has the ability to actually calculate the energy performance of your design (note: I said **calculate**, not **estimate**).

- Revit is utterly worthless when it comes to cost estimating, Facility Management, Project Planning, and numerous other BIM uses throughout the lifecycle of a building. Sure, you can hack your way through them, but for any of these, Revit is not the intended tool.

2. Loss of data.

Interesting subject. And we can go on and on about it, but it isn't true. Which I will prove during my class, and later on in this handout. Again, it's not an issue concerning the format, it's an issue concerning implementation.

All data can be transferred using IFC. Period.

3. Loss of intelligence.

When people say "loss of intelligence", they often mean "loss of parametric constraints". When importing IFC's in Revit you lose parametric capabilities: the object no longer responds to parametric changes.

Let's distinguish between the two:

- Loss of intelligence.
This happens when "smart" functionality in Revit no longer functions properly. Such as Doors / Windows that no longer "know" what Room they belong to, Ducts that can no longer be used for Pressure Loss reports, that sort of thing.
To be fair, that doesn't happen a lot. And when it does, it's usually because the import isn't properly managed.
- Loss of parametric constraints.
This is a problem in the current importer. But only if you actually want to change the imported objects. This should be questioned in the first place. Why would you want to?
But for the sake of argument: let's say you would. In this case, again, it's an implementation issue. **Not** an IFC issue. This too will be proven during this class.

4. Bi-directional links.

Really? Here's a challenge: show me one (1) bi-directional link that does exactly what you want, the way you want it.

Should you use IFC?

So in conclusion: none of the "traditional" reasons to not choose IFC are valid.

But that's no reason to proclaim "Not using Open BIM means you're not doing BIM at all".

That too is bullocks. There is no explicit need to use IFC. With one big remark:

If you can choose your partners based on their quality and they happen to all use the same software? Cool.

There's no need to use IFC if your full project scope (including Maintenance and future upgrades, extensions and renovations) can be handled in your propriety software.

But you probably don't know that for certain upon project start. So whilst IFC is not mandatory to do BIM, the **possibility** of using IFC is. Your workflow should facilitate IFC when needed.

On to more practical things

Enough with the highly theoretical stuff. On to more practical things.
Let's talk Revit. Finally!

Revit and IFC: why is it difficult

Why is the use of IFC difficult in Revit?

Surprise: it isn't.

However, think about this:

A lot of times when people come to me with questions or complaints about IFC not working, this usually happens:

Me: what settings did you apply?

Them: what settings? Why do I need settings? Why doesn't it work out of the box?

Me: how much time did you spend on getting your dwg's and pdf's to look according to your company standards?

Them: Well, like forever. That's a lot of stuff you need to worry about in Revit!

Me: Then why do you have company standards for your data exchange?

Short and simple: in order to work with IFC, you need to do some tweaking. You actually need to set up the export. A shocker, I know. But no worries, we'll take you through the steps to do this properly.

Generic file format

One thing you need to remember is that IFC is a generic file format. It's supposed to work with all sorts of software, varying from Revit to Archicad and even Sketchup. This inevitably means that you need to explain how you want the translator between Revit and IFC to interpret your model.

There are a few things you need to manage before you can successfully process an IFC:

- Geometry translation. In other words: if you model a Stair in Revit with the Stairs tool, it should be translated to IfcStair. But if you

model it as a Generic Model, it should be translated to IfcStair too. And the other way around, an AirTerminalBoxType is something completely different than an AirTerminalType. The first one is part of the Ductwork (Duct Fitting in Revit). The latter is similar to a Revit Air Terminal.

- Property translation. Revit objects have properties. So do IFC objects. The Dutch Revit User Group standards describe the resistance of a material or object towards catching fire as “brandklasse”. The IFC counterpart is “SurfaceSpreadOfFlames”. You need to tell IFC that.

Classes vs. Categories

Most important thing to realize is that both Revit and IFC have layers to determine what kind of object you’re looking at. For Revit, the layers are the Categories, as defined within the Object Styles. For IFC, the layers are the Classes.

In the dataset you will find an Excel file that lists all the IFC2x3 classes supported by Autodesk in Revit. In other words, there are 95 layers you can map your Revit Object Styles to. That is, for IFC2x3. I haven’t found a similar table for IFC4 yet.

IfcActuatorType	+ Air Terminals
IfcAirTerminalBoxType	+ Cable Tray Fittings
IfcAirTerminalType	+ Cable Trays
IfcAirToAirHeatRecoveryType	+ Casework
IfcAlarmType	+ Ceilings
IfcAnnotation	+ Columns
IfcBeam	+ Communication Devices
IfcBoilerType	+ Conduit Fittings
IfcBuildingElementProxy	+ Conduits
IfcBuildingStorey	+ Curtain Panels
IfcCableCarrierFittingType	+ Curtain Systems
IfcCableCarrierSegmentType	+ Curtain Wall Mullions
IfcCableSegmentType	+ Data Devices
IfcChillerType	+ Detail Items
IfcCoilType	+ Doors
IfcColumnType	+ Duct Accessories
IfcCompressorType	+ Duct Fittings
IfcCondenserType	+ Duct Insulations
IfcControllerType	+ Duct Linings
IfcCooledBeamType	+ Duct Placeholders
IfcCoolingTowerType	+ Ducts
IfcCovering	+ Electrical Equipment
IfcCurtainWall	+ Electrical Fixtures
IfcDamperType	+ Entourage
IfcDistributionChamberElementType	+ Fire Alarm Devices
IfcDistributionControlElement	+ Flex Ducts
IfcDistributionElement	+ Flex Pipes
IfcDistributionFlowElement	+ Floors
IfcDoorType	+ Furniture
IfcDuctFittingType	+ Furniture Systems
IfcDuctSegmentType	

Image 10 A selection of supported IFC2x3 classes in Revit vs. some Revit Object Styles

So the logical thing to do first is tie a Revit Object Style to an IFC Class. Most of the major Revit Object Styles have their IFC counterpart. Walls are translated to IfcWall, Floors to IfcSlab, Doors to IfcDoor, and so on. However there are some parts that don't really fit nicely. Most of these are part of the MEP domain.

For instance the Revit Category "Mechanical". This can be just about any mechanical component of any HVAC system. In IFC there's a more refined classification.

Not only are there more different Classes, there's also predefined types within a certain Class.

For instance the class IfcChiller (cooling equipment), has the following predefined types:

- AIRCOOLED: Air cooled chiller.
- WATERCOOLED: Water cooled chiller.
- HEATRECOVERY: Heat recovery chiller.
- USERDEFINED: User-defined chiller type.
- NOTDEFINED: Undefined chiller type.

The general rule of thumb: if you want to export something from Revit to IFC, make sure you have a (Sub) Object Style for every IfcClass you're going to use. If needed, you could also set an IFC Class Type.

The same also applies the other way around. The Class IfcCovering has several types:

- CEILING: the covering is used to represent a ceiling
- FLOORING: the covering is used to represent a flooring
- CLADDING: the covering is used to represent a cladding
- ROOFING: the covering is used to represent a roof
- INSULATION: the covering is used to insulate an element for thermal or acoustic purposes.
- MEMBRANE: an impervious layer that could be used for e.g. roof covering (below tiling - that may be known as sarking etc.) or as a damp proof course membrane
- SLEEVING: the covering is used to isolate a distribution element from a space in which it is contained.
- WRAPPING: the covering is used for wrapping particularly of distribution elements using tape.
- USERDEFINED: user defined type of covering
- NOTDEFINED: undefined type of covering

Unfortunately it's not yet possible to differentiate import on Ifc Class Types.

Mapping Tables

So how does one define the translation from Revit Categories to IFC Classes? The geeky way off course: with simple txt-files. No flashy User Interface, just plain old notepad.

Let's settle the export first. Go to R > Export > All the way down to Options > IFC Options

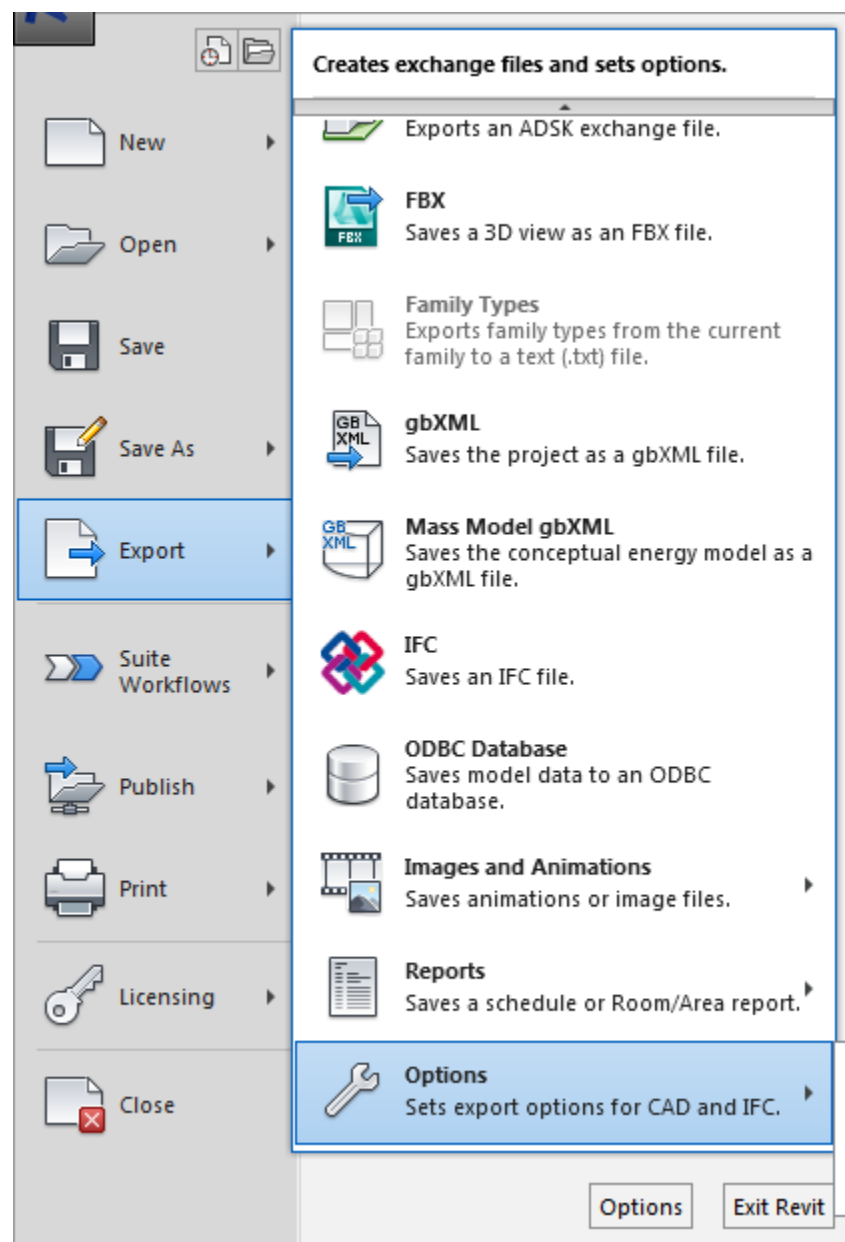


Image 11 Find the IFC Export Options here (you need to scroll down!)

This will open up a popup window where you will see three columns:

- Revit (Sub) Category
- IFC Class Name
- Type (the previously mentioned type definitions within a Class).

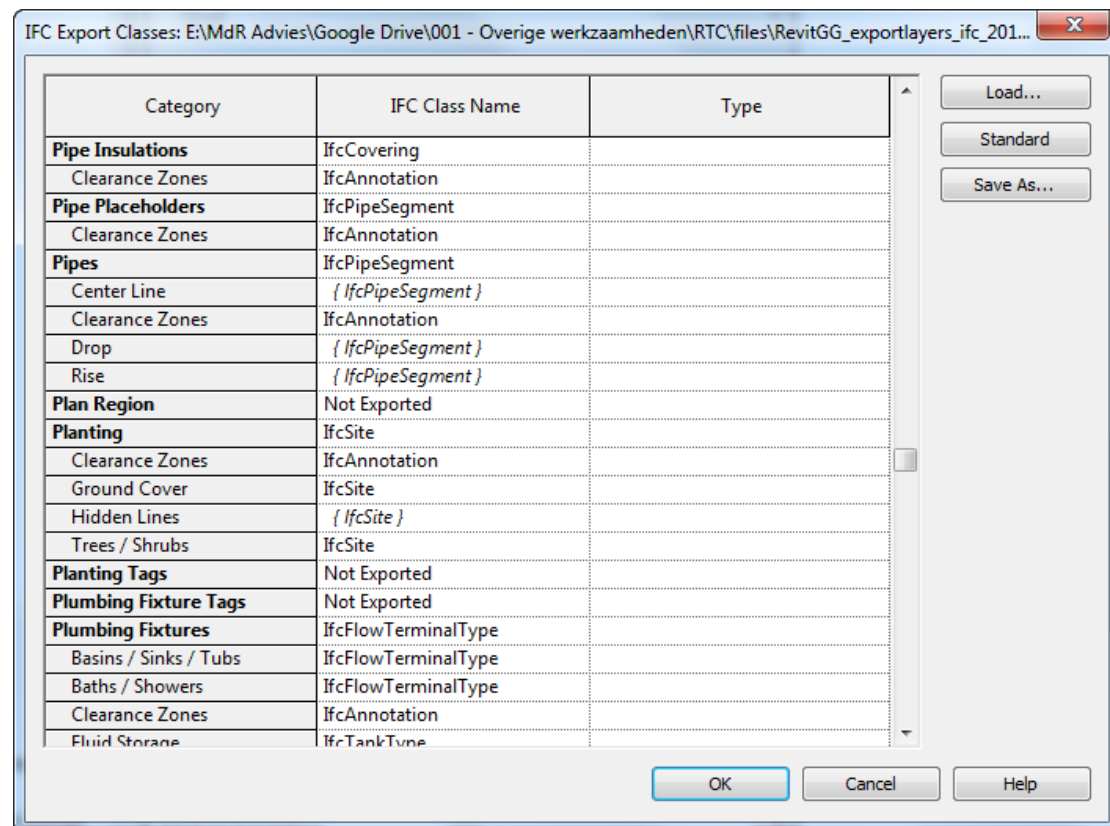


Image 12 IFC Export mapping

Basically what you need to do is fill these out. Afterwards, use Save As to store it externally for later re-use.

Same basically goes for the import side. Under Big R > Open > IFC Options to open a similar window for the import settings.

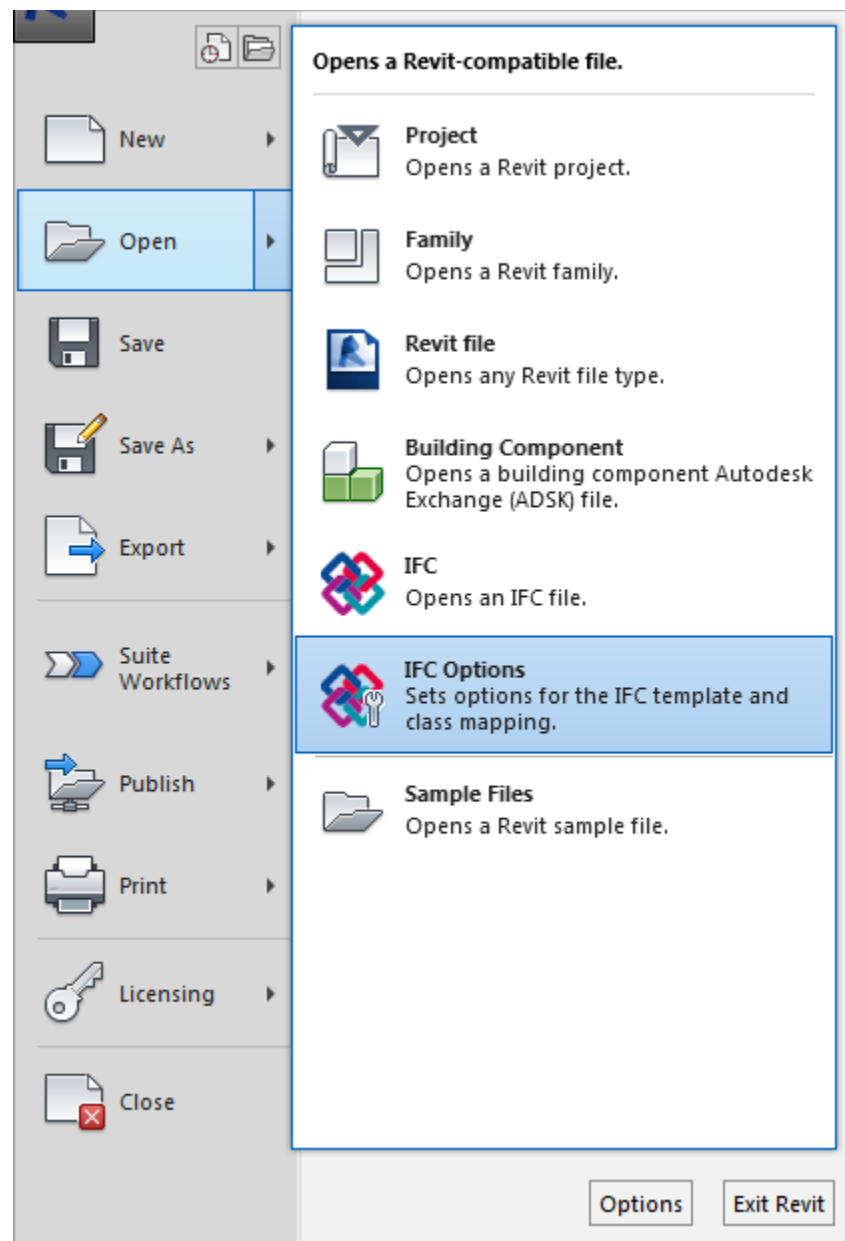


Image 13 Find the IFC Import Options here

There's a tiny problem with the standard import file shipped with Revit. 86% of the IFC Classes is mapped to Generic Models. That's like taking a 3D DWG, put 86% of all Layers to layer 0 and import that into Revit. Then wonder about why it works so poorly...

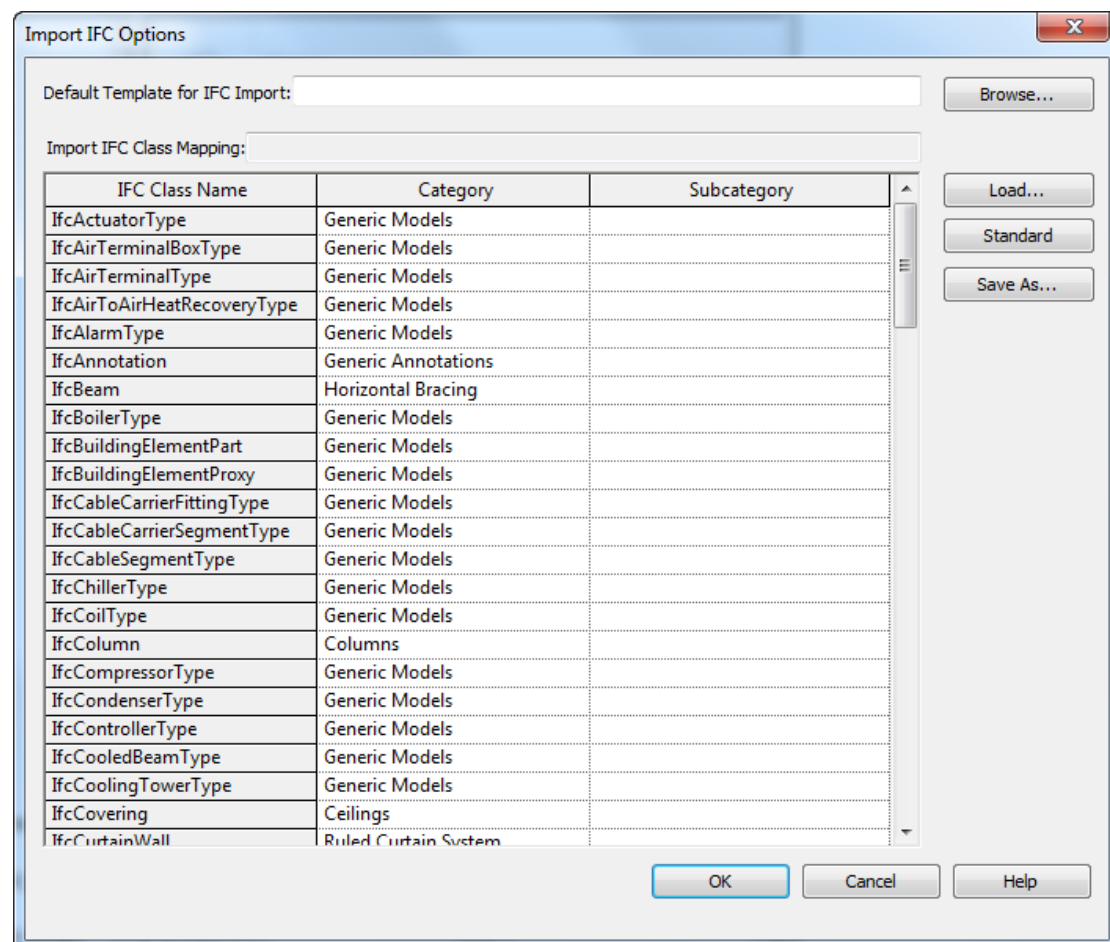


Image 14 Lots of Generic Models in the standard import table.

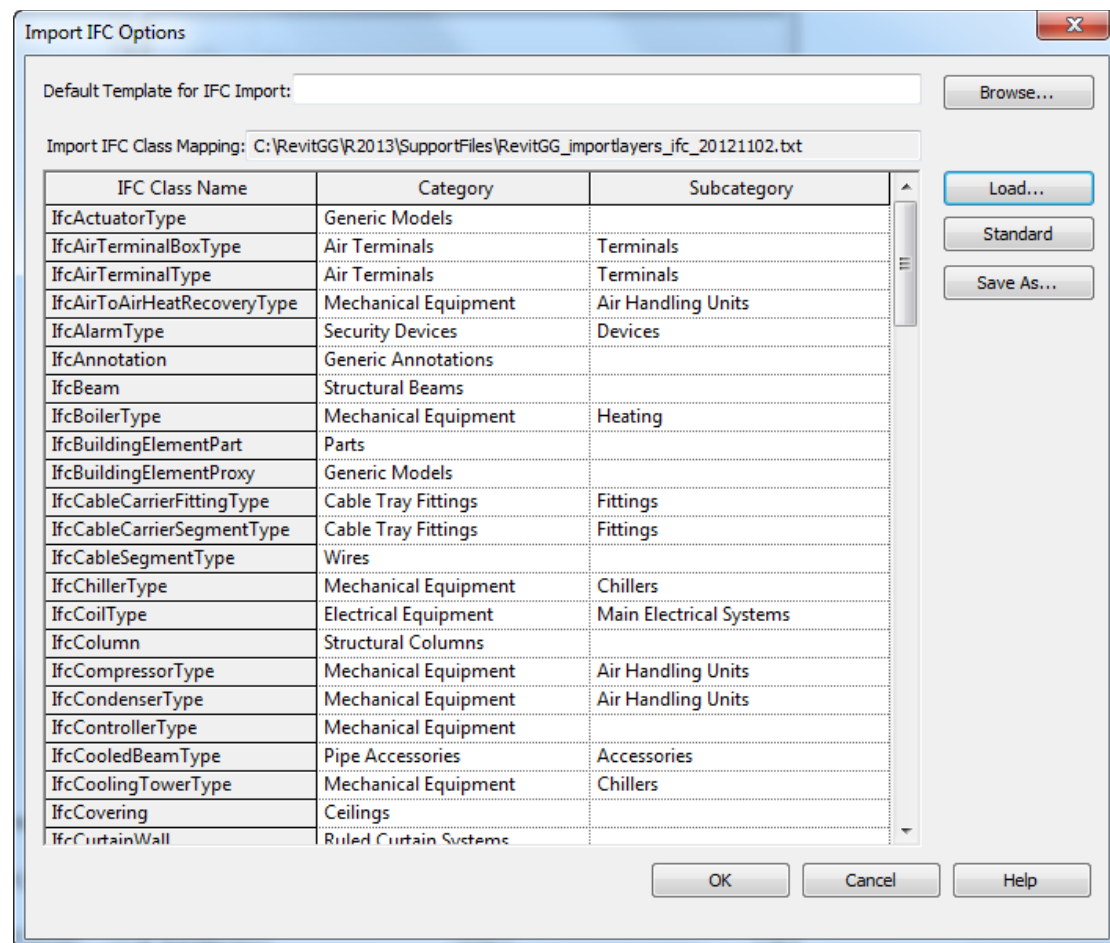


Image 15 Same table but properly filled out

In the Dataset I added two txt-files that give you an idea of a proper mapping implementation. Consider yourself warned though: there is **no** guarantee these mappings actually work. They have been tested to the best of my knowledge, but since there is no documentation whatsoever on the subject, it's just my best guess.

Furthermore: you will of course need to match the Revit Object Styles to the properties in the table.

Manual overrides

There will be times that a certain Object needs to be mapped to another IFC Class then the general export mapping table tells it to. Think situations where you have a unique type of family modelled as a Generic Model.

Besides the general mapping table you have the ability to use 2 Shared Parameters provided by Autodesk:

IfcExportAs: Overrides or sets an object specific IFC Class to which the object will be mapped.

IfcExportType: Assigns a specific Type (Subcategory) within the specified IFC Class.

For instance:

You modelled an Air Cooled Chiller. Add the two parameters to the family and set them accordingly:

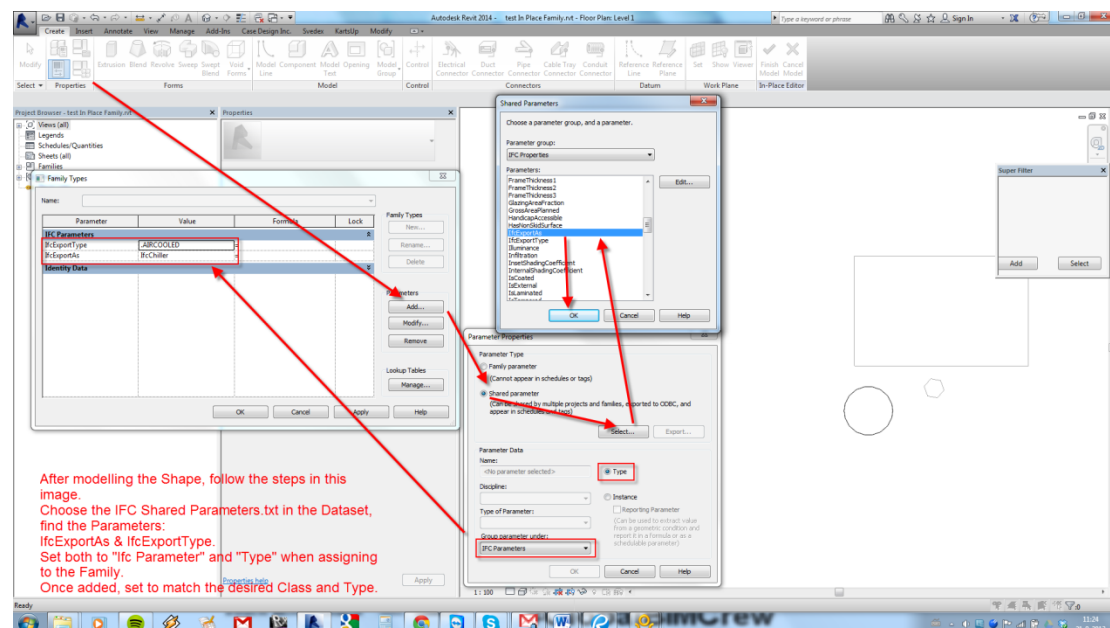


Image 16 Assigning and setting up the IfcExportAs and IfcExportType parameters

However, there are some noticeable limitations to this workflow:

1. System Families will generally default to the export settings in the mapping table or, even worse, to hardcoded settings that you cannot touch at all. Doesn't matter if you use a "normal" or "In Place" family.
This means that creating an In Place Ceiling and having it export to something else (even to the CEILING type will not work unless you do this in the mapping table).
2. In Place Families **must** have an IfcExportAs parameter assigned or they will default to BuildingElementProxy.
3. To define a certain type you can either use the IfcExportType parameter OR add the type to the IfcExportAs parameter with the following syntax: IfcClass.Ifctype.

4. The IfcExportAs parameter can be assigned to either the project file (or template) or to the family. You'd probably want a mix of these two. Generic Models in general have it prepopulated in the family, for other categories that only marginally need to be adjusted, having it in the project template will do.

In the dataset you will find a test file with resulting IFC to examine. The results of the different settings applied are shown in the image below.

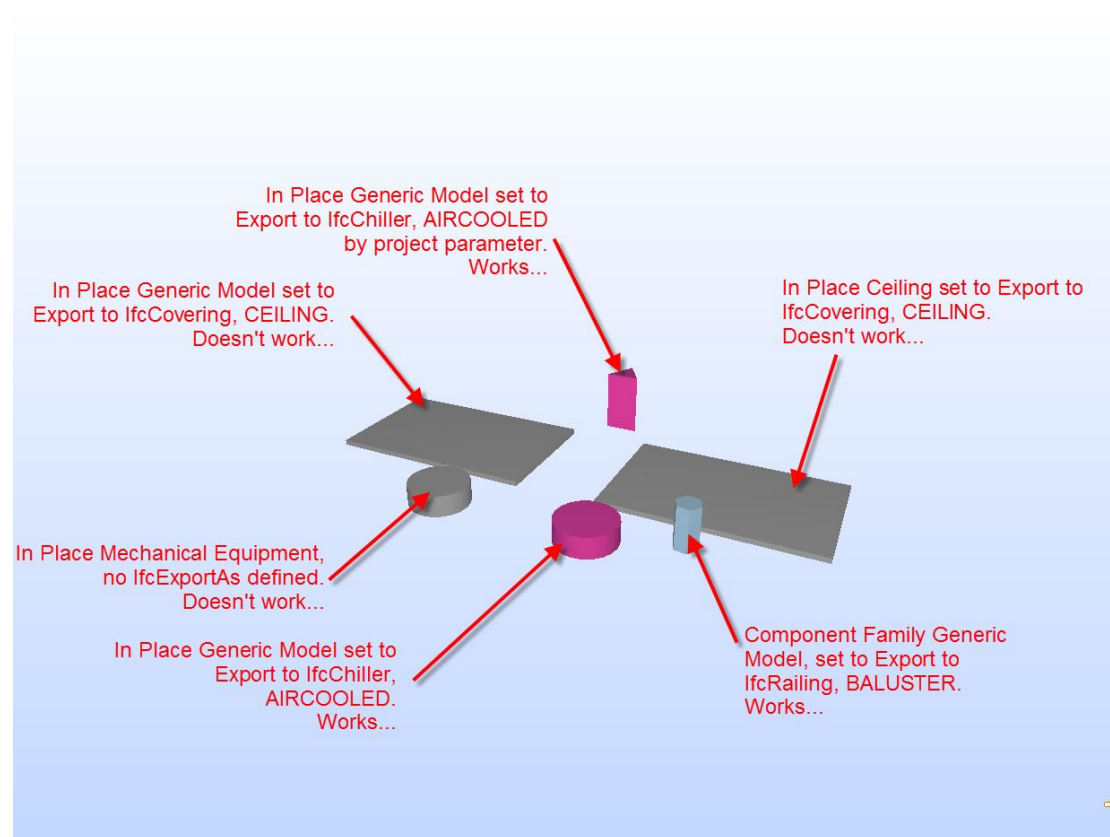


Image 17 several use-cases of the IfcExportAs parameters.

Setting up IFC Layers

The IFC File format has a dwg-like layer structure, see the image below. This will probably make sense to US residents, but not to me. I would like this layer definition to match our national (or even company) standards.

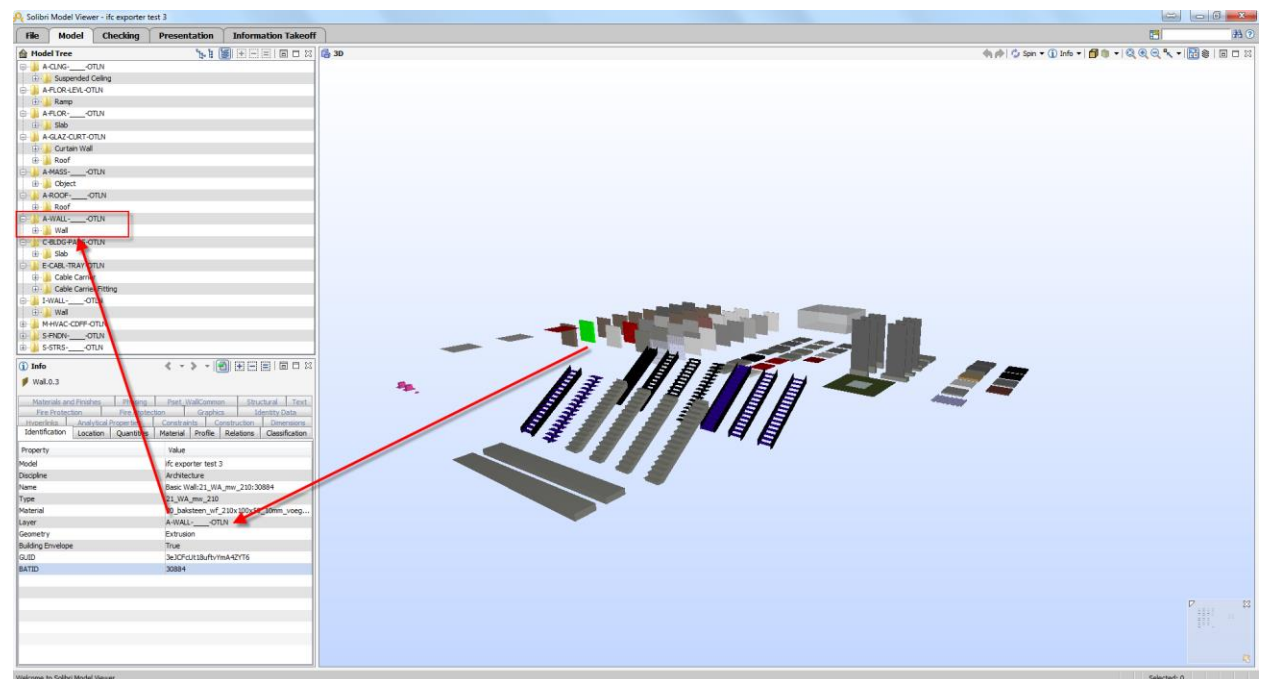


Image 18 Default IFC layer definition.

Luckily, this is fairly simple to manage. That is, once you found the trick:

1. Find the Revit.ini file. Default location is
 %APPDATA%\Autodesk\Revit\Autodesk Revit 2014\.
2. Open the file with Notepad, and add the following lines:
 Under [Directories] add:
 ExportLayersNameDGN=xxx.txt (with xxx being the full path and filename of your desired dwg mapping table)
 Under [Export] add:
 UseVersion2012DGN=1
3. Export model to IFC to find it mimics the dgn export settings. (If you were running Revit while making these changes you should restart it for the changes take effect).

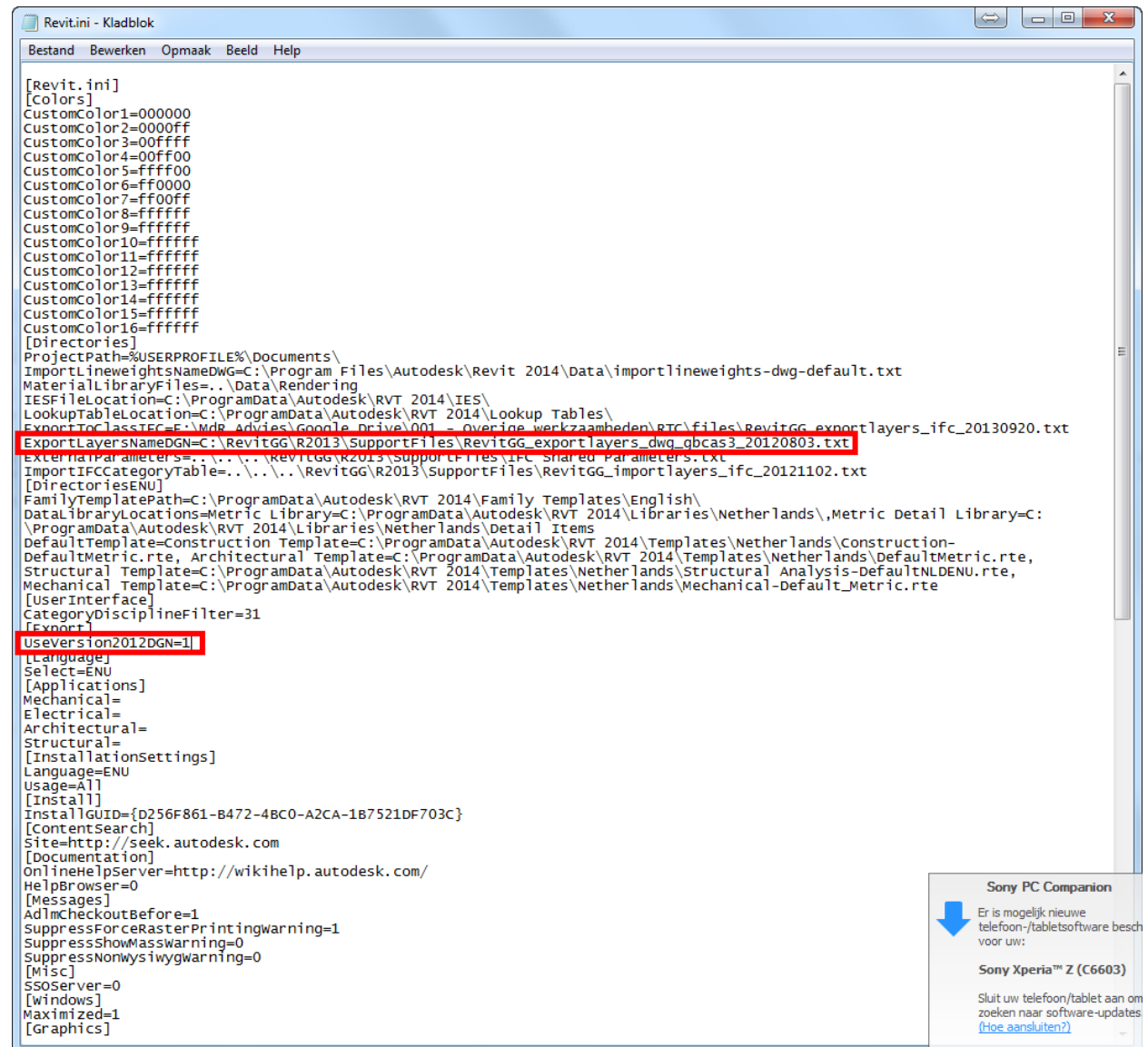


Image 19 Export settings to add or adjust in the Revit.ini file

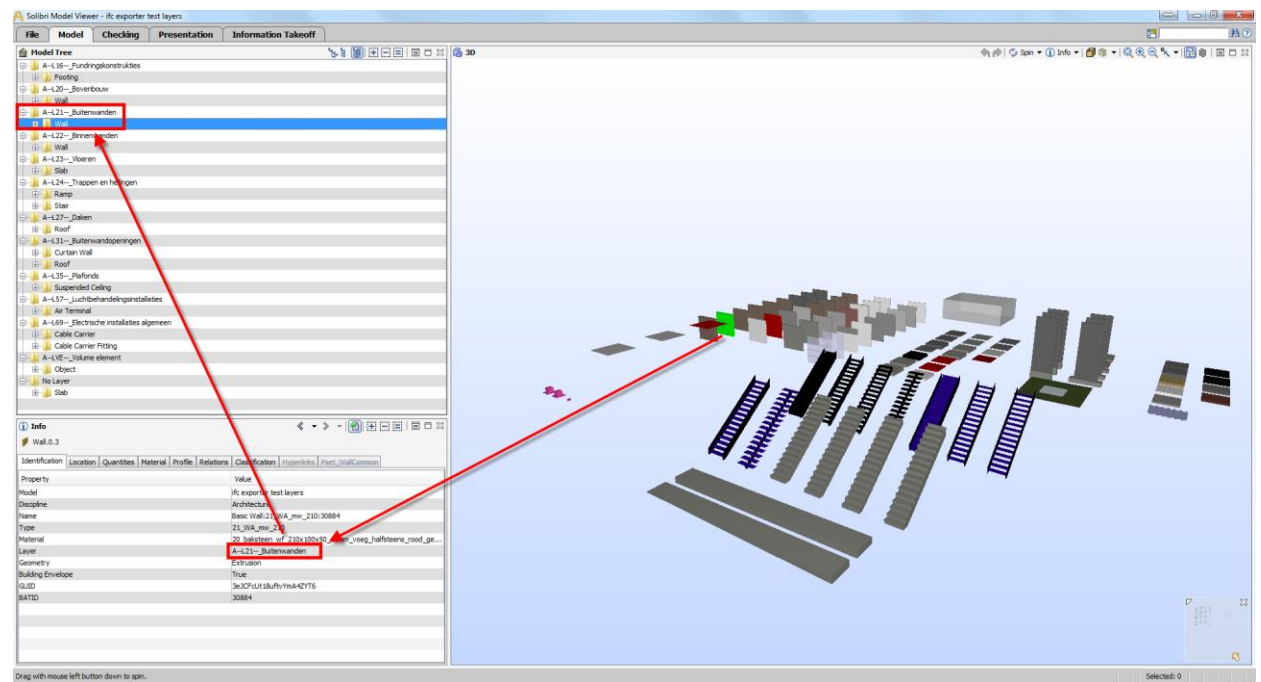


Image 20 Customized IFC layer definitions

(Additional) tips and tricks on managing IFC export/import

- Use (Sub) Object Styles for each IfcClass you need to use. Set up a mapping table to predefine the way these Object Styles are translated, for both import and export. Examples can be found in the Class Dataset.
 - Specific type enumerations can be defined in the Export mapping table or by using the IfcExportType parameter. This will only work for Export. Importing IFC's does not support the use of IFC Types (yet).
 - Use the Open Source Exporter when handling IFC's. It's better, it gets updated each month instead of once a year and the support is awesome. Find it here:
<http://sourceforge.net/projects/ifcexporter/>
 - Use the IFC2x3 technical site to find listings of available IFC Classes and Types. Find it here: <http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm>. If you don't know what to choose, go to the Architectural Diagram and start from the top.
 - Add the IfcExportAs and IfcExportType parameters to your project template and if necessary in your families. Make sure it's classified as a Type Parameter and put in the Ifc Parameter group. Otherwise it won't work.
4. IFC uses layers for classification. It defaults to the hardcoded Revit layers you might know from an OOTB dwg export. If you want to change that, add two lines of code to your Revit.ini file.
- Under [Directories] add:
ExportLayersNameDGN=xxx.txt (with xxx being the full path and filename of your desired dwg mapping table)
- Under [Export] add:
UseVersion2012DGN=1
- When working with complex families (for instance Doors and Windows), use Shared nested components. This way the different parts will be mapped to different Ifc Classes as separate objects.
 - To **not** export a single object type, use the IfcExportAs and type in "DontExport" (without the " "). The object type will be ignored when exporting.

The cool stuff

There have been three developments in the last few months that offer insights about the true potential of IFC as a carrier of information.

Besides the basic setup you need to go through before working with Revit, these three will be the main focus of the class itself.

All of these developments relate to the way data is transferred from Revit into IFC. Before getting into this, it's important to understand the difference between the way Revit handles parameters, and the way IFC does.

Each IFC Class (IfcWall, IfcFooting) has a set of properties defined by BuildingSmart in the IFC Scheme. These properties are placed in "Property Sets", or "Psets". For Walls, the predefined PSet is Pset_WallCommon. These Psets contain a certain amount of predefined properties which can be roughly compared to Revit parameters.

Name
Reference
AcousticRating
FireRating
Combustible
SurfaceSpreadOfFlame
ThermalTransmittance
IsExternal
ExtendToStructure
LoadBearing
Compartmentation

Image 21 Predefined properties of IfcWall

However, there are a few noticeable differences with Revit:

1. IFC Psets do not know the concept of Parameter Groups like in Revit. In fact, a PSet is a single Group. But more on that later...

When exporting to IFC from Revit, you have the ability to check "Export internal Revit Property Sets", see image.

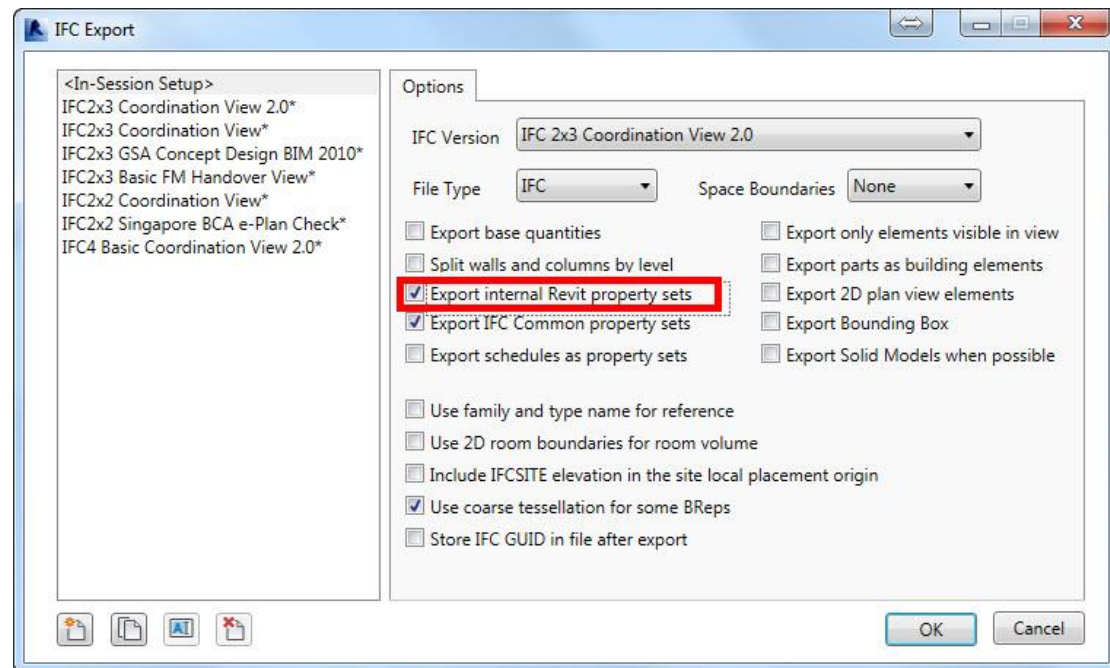


Image 22 Exporting Revit internal Property Sets through the Open Source Exporter

Selecting this checkbox will result in an IFC file where all Revit Parameter Groups are exported as separate Property Sets in IFC.

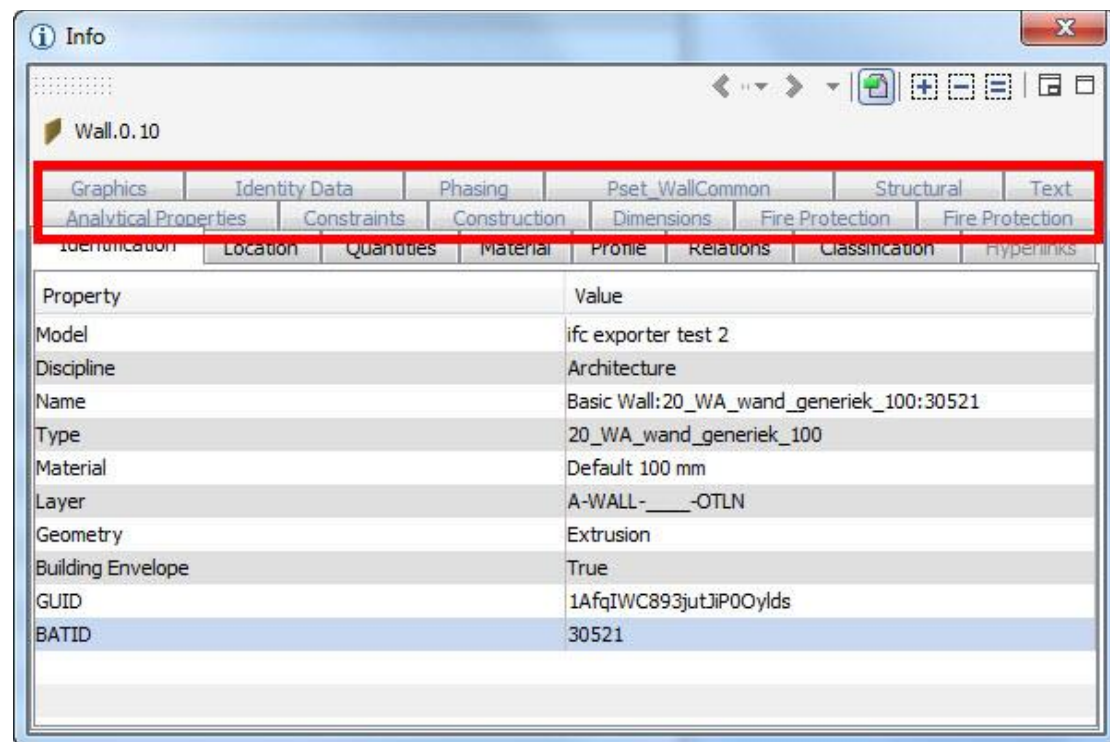


Image 23 Internal Revit Property Sets in IFC

This, off course, isn't really helpful. Why not? Well, simply because the organization of parameters in the IFC becomes dependent of the organization in Revit. While in Revit, this is merely a User Interface thing, in IFC this will cause problems.

Let's say I have some software for Facility Management that can read and process IFC parameters automatically. Chances are this software will need to know where to find these parameters and how they are named. This means that this needs to be known and managed in Revit. Only problem is: by the time the IFC gets to the FM company, chances are that it's a "take it or leave it" kind of handover. They will have to jump through some severe hoops to make it work.

Let's look at some IFC code to see what's happening here:

If we look at the Wall from image 5, it's defined in text row #170:

```
#170= IFCWALLSTANDARDCASE('1AfqIWC893jutJiP0Oylds',#41,'Basic
Wall:20_WA_wand_generiek_100:30521',$','Basic
Wall:20_WA_wand_generiek_100:1384',#139,#166,'30521');
```

The wall has some properties, of which the definition can be found in line #200 to #237. Some of them concern Fire:

```
#200= IFCPROPERTYSINGLEVALUE('brandwerend',$,IFCBOOLEAN(.T.),$);  
#201= IFCPROPERTYSINGLEVALUE('fire_rating',$,IFCTEXT('30'),$);  
#225= IFCPROPERTYSINGLEVALUE('ontvlambaar',$,IFCBOOLEAN(.F.),$);  
#235= IFCPROPERTYSINGLEVALUE('Fire Rating',$,IFCTEXT(''),$);
```

These are defined in three distinct Property Sets:

```
#252= IFCPROPERTYSET('1AfqlWC893jutJw4Oylds',#41,'Fire Protection',$,(#200,#201));  
#276= IFCPROPERTYSET('1Sps1bgmr0zeLnc8098PCC',#41,'Fire Protection',$,(#225));  
#280= IFCPROPERTYSET('1Sps1bgmr0zeLncB098PCC',#41,'Identity  
Data',$,(#232,#233,#234,#235,#236,#237));
```

None of them belong to the standard Pset_WallCommon defined by BuildingSmart.

A second problem is that the IFC is riddled with stuff you don't need...:

```
#207= IFCPROPERTYSINGLEVALUE('Base is Attached',$,IFCBOOLEAN(.F.),$);  
#208= IFCPROPERTYSINGLEVALUE('Base Offset',$,IFCLENGTHMEASURE(0.),$);  
#209= IFCPROPERTYSINGLEVALUE('Location Line',$,IFCIDENTIFIER('Core Face:  
Exterior'),$);  
#210= IFCPROPERTYSINGLEVALUE('Related to Mass',$,IFCBOOLEAN(.F.),$);  
#211= IFCPROPERTYSINGLEVALUE('Room Bounding',$,IFCBOOLEAN(.T.),$);  
#213= IFCPROPERTYSINGLEVALUE('Top is Attached',$,IFCBOOLEAN(.F.),$);  
#217= IFCPROPERTYSINGLEVALUE('Enable Analytical Model',$,IFCBOOLEAN(.F.),$);  
#226= IFCPROPERTYSINGLEVALUE('Coarse Scale Fill Color',$,IFCINTEGER(8421504),$);  
#227= IFCPROPERTYSINGLEVALUE('Coarse Scale Fill Pattern',$,IFCLABEL('Solid fill'),$);  
#230= IFCPROPERTYSINGLEVALUE('Wrapping at Ends',$,IFCIDENTIFIER('None'),$);  
#231= IFCPROPERTYSINGLEVALUE('Wrapping at Inserts',$,IFCIDENTIFIER('Do not  
wrap'),$);
```

These are all things that nobody cares about when exchanging models.

Cool feature 1: Parameter mapping to IFC parameters.

As part of the Dutch Revit Standards, the Dutch Revit User Group aimed at making the handling of IFC with Revit better. More controllable. What we've seen thus far in this handout is that it's mostly an "all-or-nothing" deal. We want the user to be in charge of the data they transfer (and receive).

So the first thing we did was utilizing the properties currently present in IFC. As we established earlier, most IFC Classes have the ability to carry a limited amount of user defined properties for that specific type of object (besides the geometry and general properties).

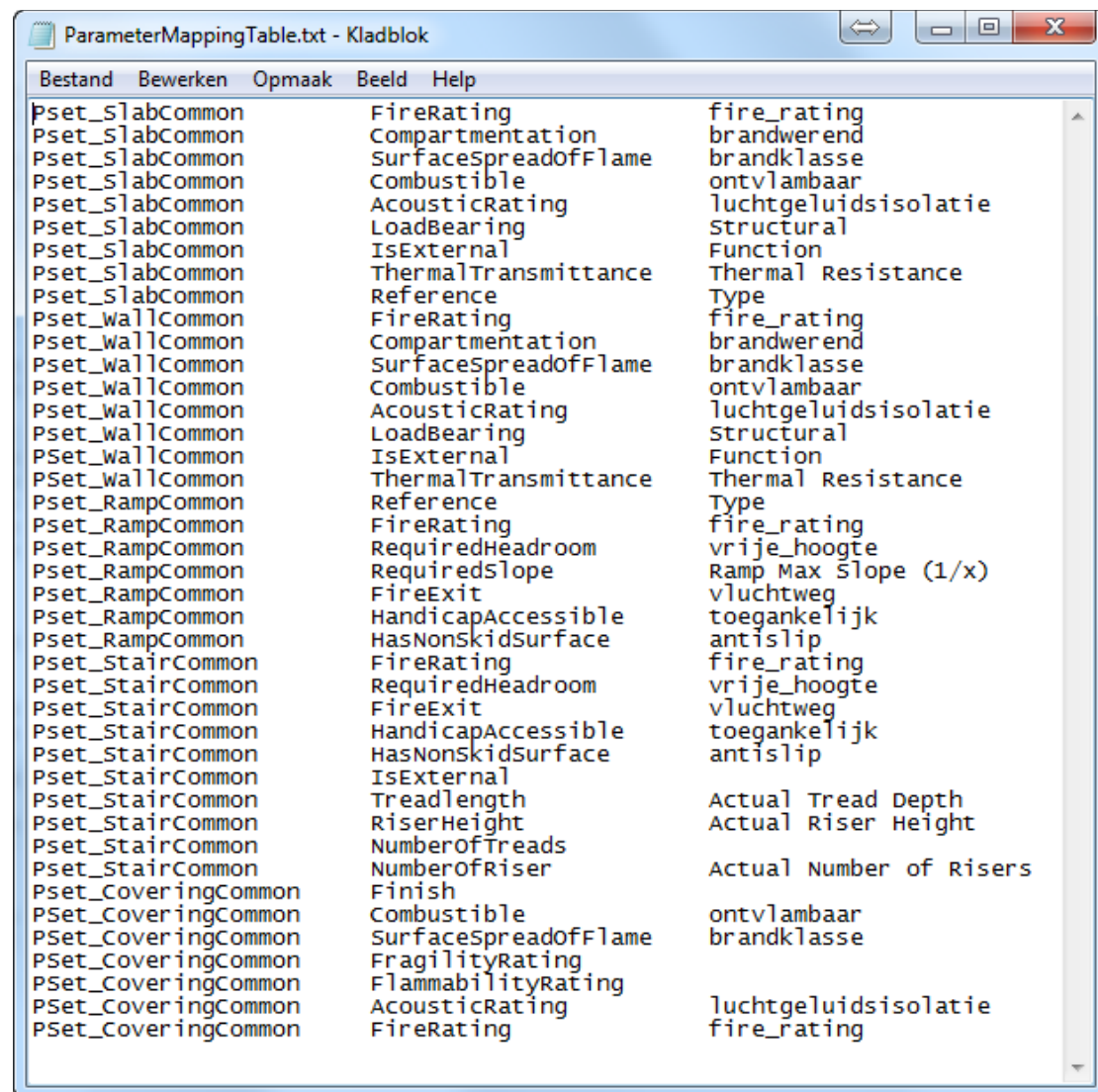
However, current Revit export doesn't support using all these parameters. And, for the parameters it does support, you'd need to use the Autodesk IFC Shared Parameters list. This is kind of a problem, since the Dutch RUG has its own list of standardized Shared Parameters (as do most companies).

So we asked Tom Pesman (<http://tnux.net/>) to create the ability to map your company Shared Parameters to the default IFC Properties. The User Interface of this added functionality is horrific (actually, it's not there), but it works. If you want pretty menus, have a go at it... This addition was published in the Autodesk IFC Open Source Exporter back in June. So the source code is available to the public. But for us, since all of these developments are on a voluntary basis, the end result is the only thing that matters.

How does it work? Well, fairly easy and straightforward actually. The entire User Interface consists of a txt-file which sits in the Open Source Exporter install directory. This txt-file has a simple format:

Property set <tab> Revit parameter <tab> IFC property

That's it. Applied to the Dutch Revit Standards, this is what it looks like:



Bestand	Bewerken	Opmaak	Beeld	Help
Pset_SlabCommon	FireRating	fire_rating		
Pset_SlabCommon	Compartmentation	brandwerend		
Pset_SlabCommon	SurfaceSpreadofFlame	brandklasse		
Pset_SlabCommon	Combustible	ontvlambaar		
Pset_SlabCommon	AcousticRating	luchtgeluidsisolatie		
Pset_SlabCommon	LoadBearing	Structural		
Pset_SlabCommon	IsExternal	Function		
Pset_SlabCommon	ThermalTransmittance	Thermal Resistance		
Pset_SlabCommon	Reference	Type		
Pset_WallCommon	FireRating	fire_rating		
Pset_WallCommon	Compartmentation	brandwerend		
Pset_WallCommon	SurfaceSpreadofFlame	brandklasse		
Pset_WallCommon	Combustible	ontvlambaar		
Pset_WallCommon	AcousticRating	luchtgeluidsisolatie		
Pset_WallCommon	LoadBearing	Structural		
Pset_WallCommon	IsExternal	Function		
Pset_WallCommon	ThermalTransmittance	Thermal Resistance		
Pset_WallCommon	Reference	Type		
Pset_RampCommon	FireRating	fire_rating		
Pset_RampCommon	RequiredHeadroom	vrije_hoogte		
Pset_RampCommon	RequiredSlope	Ramp Max Slope (1/x)		
Pset_RampCommon	FireExit	vluchtweg		
Pset_RampCommon	HandicapAccessible	toegankelijk		
Pset_RampCommon	HasNonSkidSurface	antislip		
Pset_StairCommon	FireRating	fire_rating		
Pset_StairCommon	RequiredHeadroom	vrije_hoogte		
Pset_StairCommon	FireExit	vluchtweg		
Pset_StairCommon	HandicapAccessible	toegankelijk		
Pset_StairCommon	HasNonSkidSurface	antislip		
Pset_StairCommon	IsExternal			
Pset_StairCommon	Treadlength	Actual Tread Depth		
Pset_StairCommon	RiserHeight	Actual Riser Height		
Pset_StairCommon	NumberOfTreads			
Pset_StairCommon	NumberOfRiser	Actual Number of Risers		
Pset_CoveringCommon	Finish			
Pset_CoveringCommon	Combustible	ontvlambaar		
Pset_CoveringCommon	SurfaceSpreadofFlame	brandklasse		
Pset_CoveringCommon	FragilityRating			
Pset_CoveringCommon	FlammabilityRating			
Pset_CoveringCommon	AcousticRating	luchtgeluidsisolatie		
Pset_CoveringCommon	FireRating	fire_rating		

Image 24 Parameter mapping for some Revit Categories

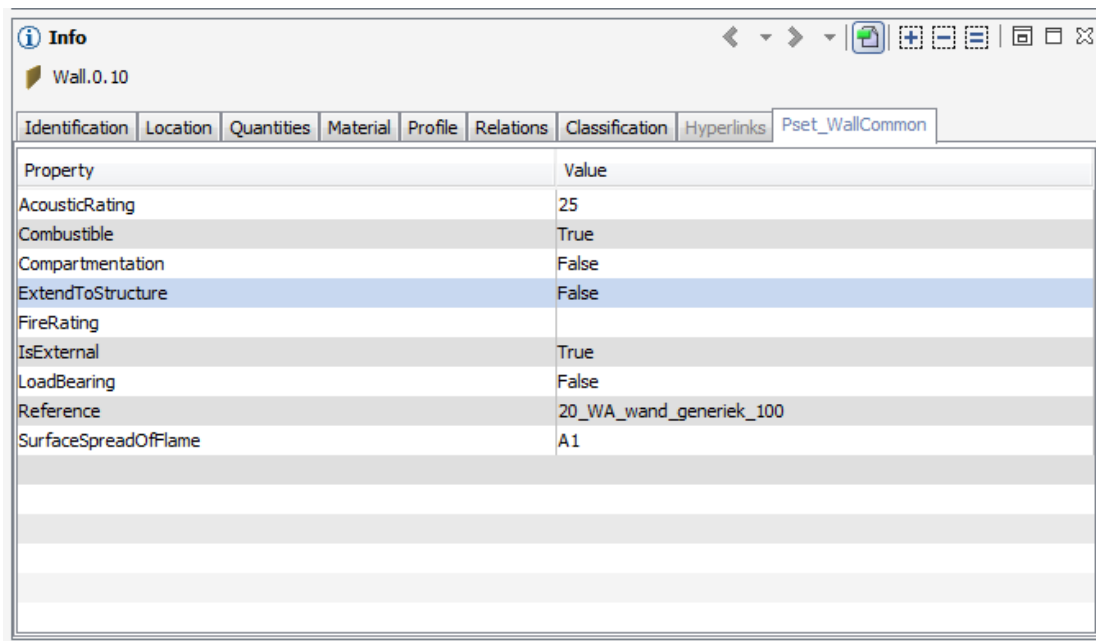
And these are the results in numbers:

File size down with 30%

Lines of code down from 36518 to 28825 lines.

But most important:

All Wall properties defined in the IFC scheme can be exported properly, without the endless amount of internal parameters.



Property	Value
AcousticRating	25
Combustible	True
Compartmentation	False
ExtendToStructure	False
FireRating	
IsExternal	True
LoadBearing	False
Reference	20_WA_wand_generiek_100
SurfaceSpreadOfFlame	A1

Image 25 Wall properties using the Open Source Exporter

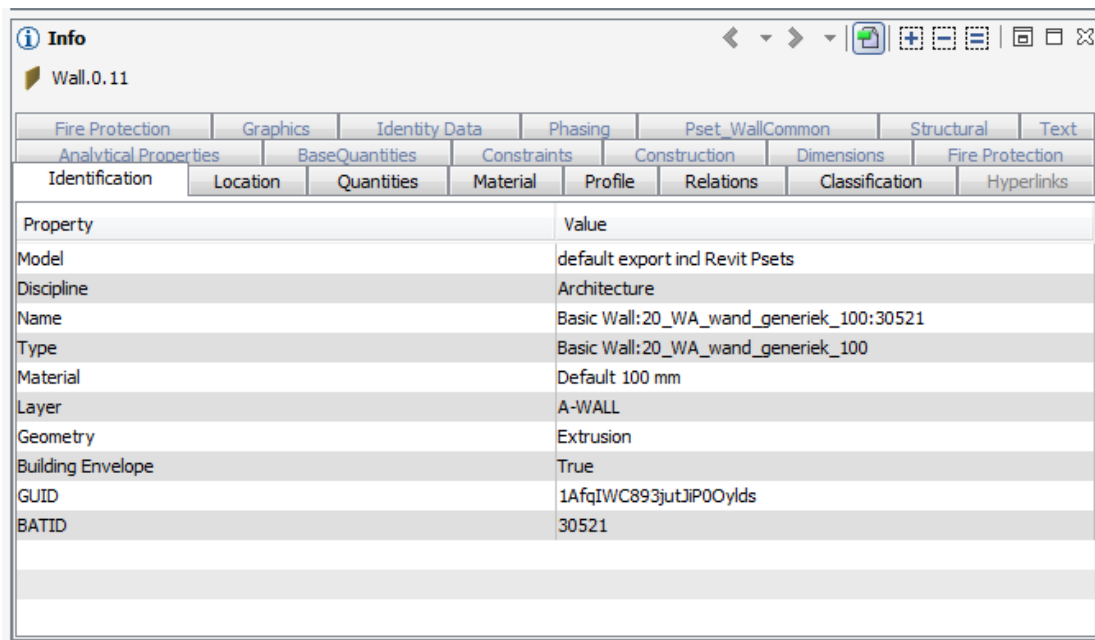


Image 26 Wall properties using the internal Revit Property Sets

Cool feature 2: Creating custom properties in IFC

Phase two is somewhat more complex. It's nice that we can now fill the standard IFC properties with Revit parameter values of our choice, but that is hardly sufficient. IFC Property Sets only define the most common parameters for any object.

What if, for instance, we'd want a Cobie data drop to IFC? At this point we're stuck with the internal Revit Property Sets, and all drizzle that comes with it. Another option is to use the newly created option in the latest version of the Open Source Exporter to export Schedules to a custom Property Set. Unfortunately this exports all schedules present in the model, even the ones you don't want. That could be solved by temporarily deleting all present schedules you don't want to export. But even then, schedules need to have unique names. So there won't be one single Property Set which is filled with relevant parameters for each object. In short: it's only slightly better...

Therefore the second step for the Dutch Revit User Group was creating a way to define custom Property Sets in the IFC and porting Revit parameters to that Property Set. This code has been provided by Wim Tas of Witas bvba, a Belgian MEP Engineering firm (www.witas.be). This gives users two advantages:

1. The ability to create lean IFC's that contain all data you want (and only that specific data).
2. The ability to create IFC's specifically tailored to a single project partner.

Think about it:

- You can now create a mapping table that lets you define exactly what you want exported, and to which IFC Class.
- For control on a single object level, there's the `IfcExportAs` parameter.
- To define what data you want exported, you can create separate parameter mapping tables for both standard IFC properties and custom additional properties.

How cool is that? So, how does this work?

Well, again, we're all function and no form. But unfortunately we're also a voluntary team that has to do this in the limited amount of spare time available. Because of that, this feature isn't fully developed as I write this. But I have been assured it will be ready prior to my presentation, so you'll just have to wait and join my class to see how it all works. In return I can promise we will be spending some time on doing a live demo and I'll add a how-to to the class dataset.

And, as the first addition, this too will be part of the Autodesk Sourceforge Open Source Project.

Cool feature 3: Parametric IFC import.

Up until now, all enhancements were made to improve the extraction of data out of the Revit model. But how about importing? How about when you need to use an IFC someone else created? Either because there was a handover downstream, because you're collaborating in a multi-disciplinary project, or because it's an old project that needs renovation (granted, the latter probably won't happen for a few years, but still...)

A major problem with Revit and IFC is that importing an IFC creates a stinking pile of useless garbage:

- All associated data with components is lost. This is not (I repeat: NOT) a limitation of IFC. It's an implementation issue. And it's intended use too. Reasoning behind this behaviour was that it would be confusing and potentially harmful to have the parameter values import whilst parametric constraints were lost. In other words: when Revit users see a dimension parameter filled, they assume that it's tied to the actual geometry. Autodesk wanted to prevent this mistake from happening and chose not to import parameter values.
- All constraints are lost. Even if we would get the parameter values back in, changing them would not affect model geometry.

This, I think, was a necessity back when Autodesk started working on IFC import. The programming environment to automatically create constraints during database driven family generation (which is what you're basically doing here) wasn't in place back then. However, things have changed. It is possible now, as will be demonstrated.

- Imported components are unique components; objects will not be grouped in Families and/or Types.

Ok, this would be less of a problem if points 1 and 2 weren't there. It's true: most imported objects are converted to single entities without any Family or Type relationships. So two completely similar objects will be two separate families when exported to IFC and re-imported in Revit. Again, would not be

such a problem if you had parameter values such as Mark and Type Mark to filter and sort your schedules upon.

Let's put this into a few nice pictures. Below is a screenshot from two sets of Wall properties.

First this is what we started with.

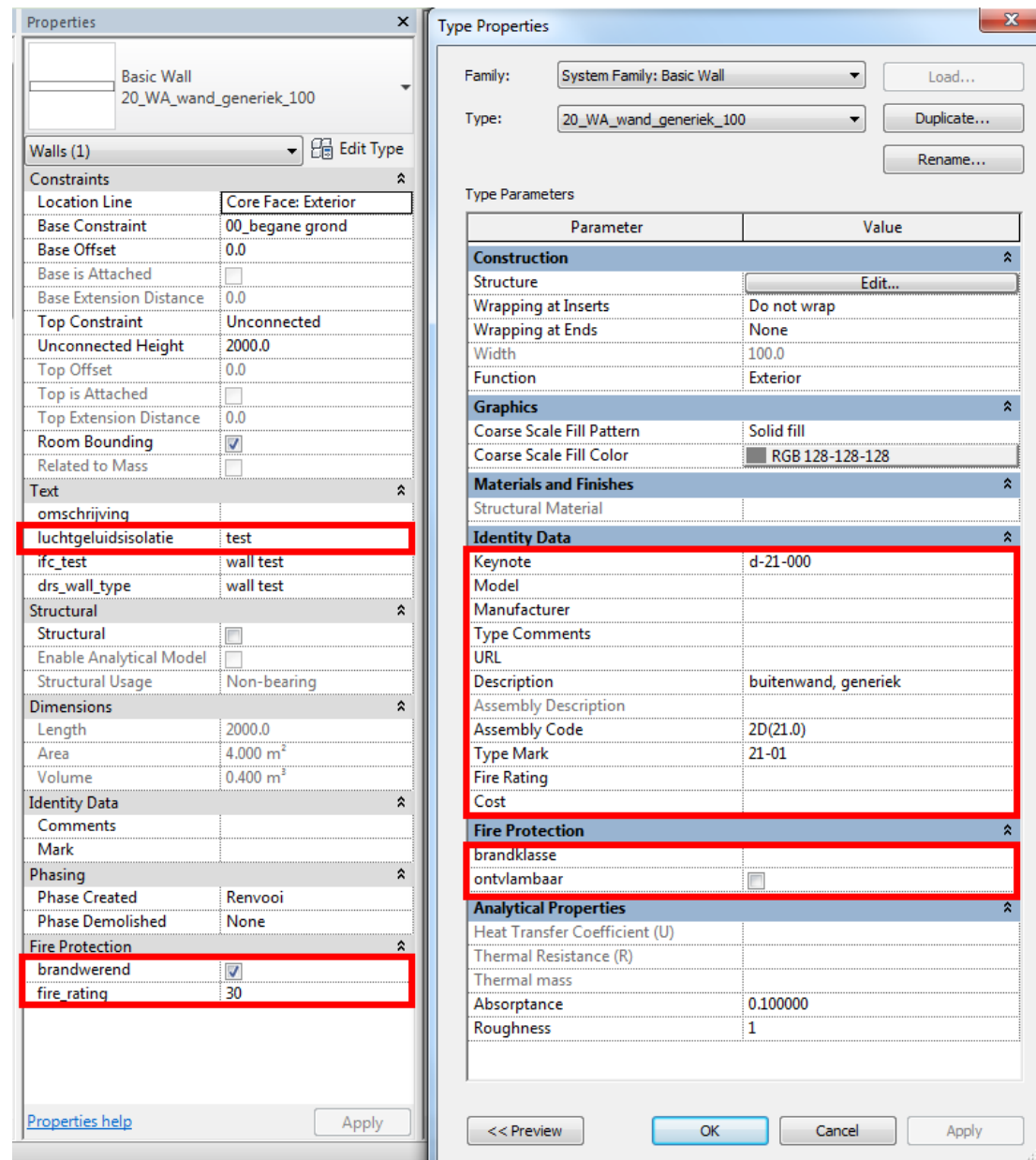


Image 27 Wall properties in Revit as native geometry

Third, the standard IFC (without the Dutch Revit Standards add-on to the Open Source Exporter) imported back into Revit using the standard Autodesk tools:

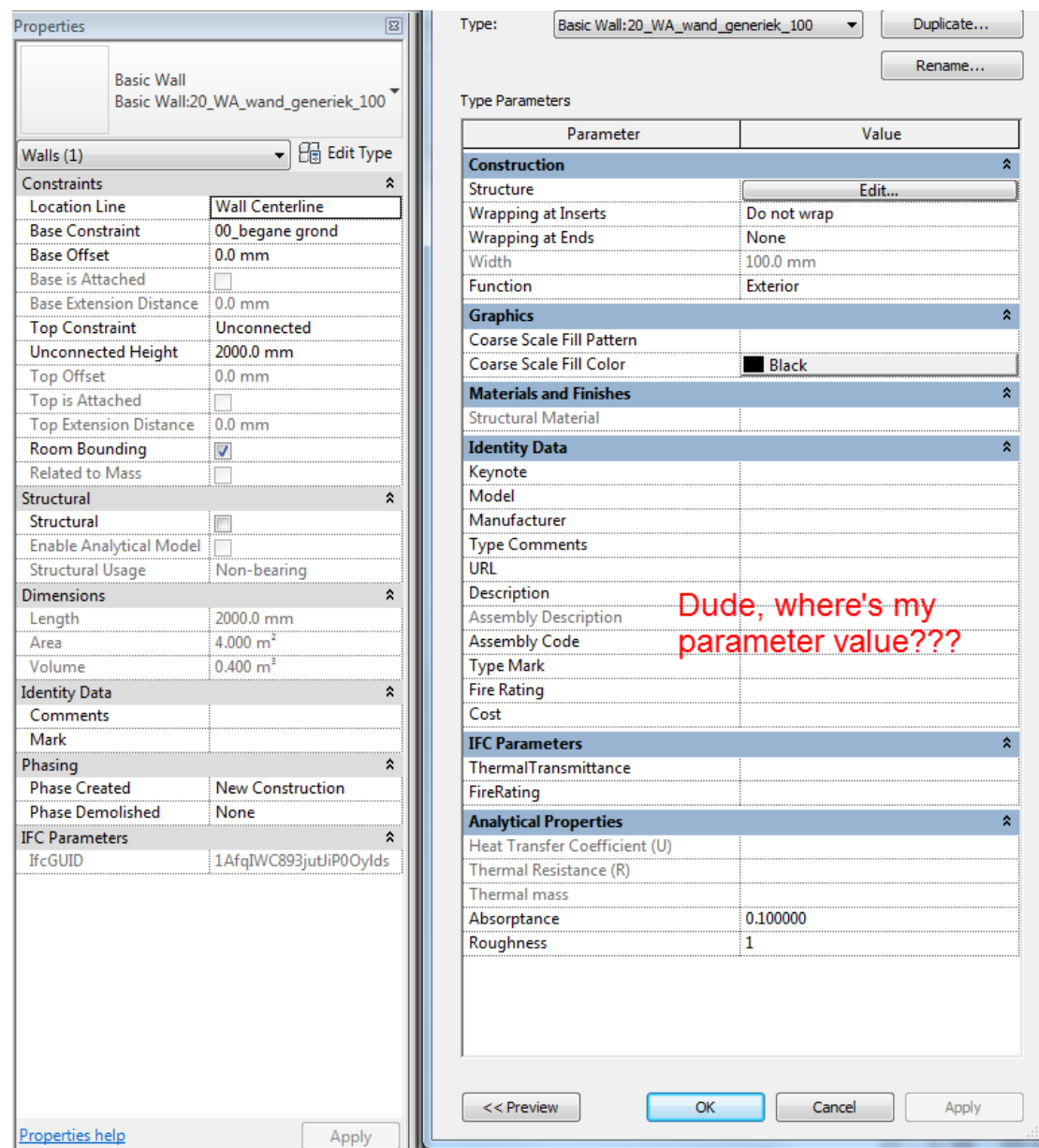


Image 28 Missing all custom Properties and no IFC properties to replace them

And finally, using the ParameterMappingTable.txt for export shows this on import:

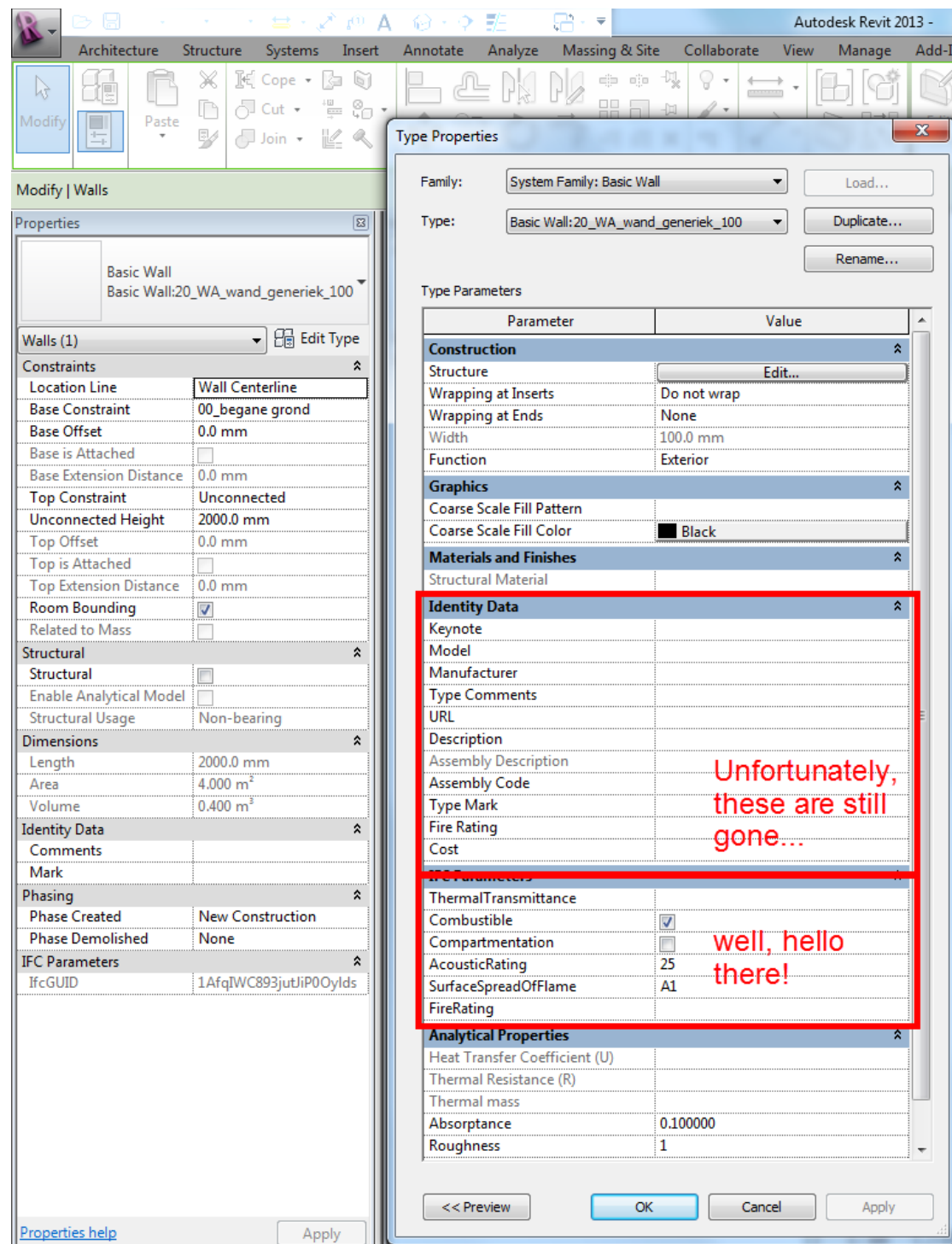


Image 29 Surprise!!! Apparently IFC parameters ARE pulled back in

IFC for Interoperability

Martijn de Riet, MdR Advies

To be honest, that was a surprise for me too. And a pleasant one. As you can plainly see, the standard IFC Properties ARE being re-imported into Revit. Not as our own Shared Parameters, but hey, that is fixable.

Not to get your hopes up too much, this is what happened to the Structural Foundations upon re-import...

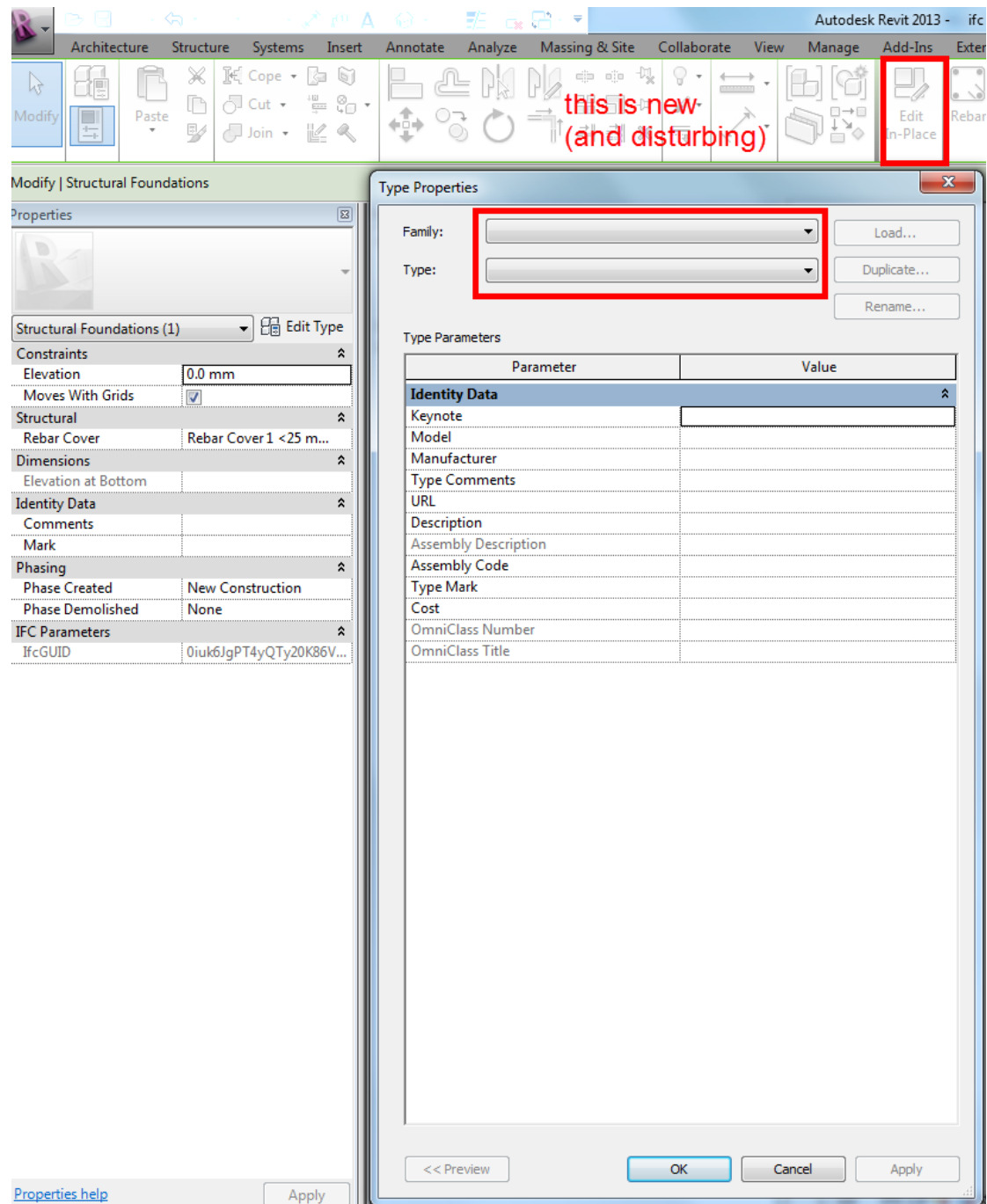


Image 30 Ouch!

But have no fear! Jon Mirtschin, IFC Guru from Geometry Gym (www.geometrygym.com), has been working on implementing a solution into his importer.

He created some structural foundations too and did some of his magic on the GG IFC Importer for Revit.

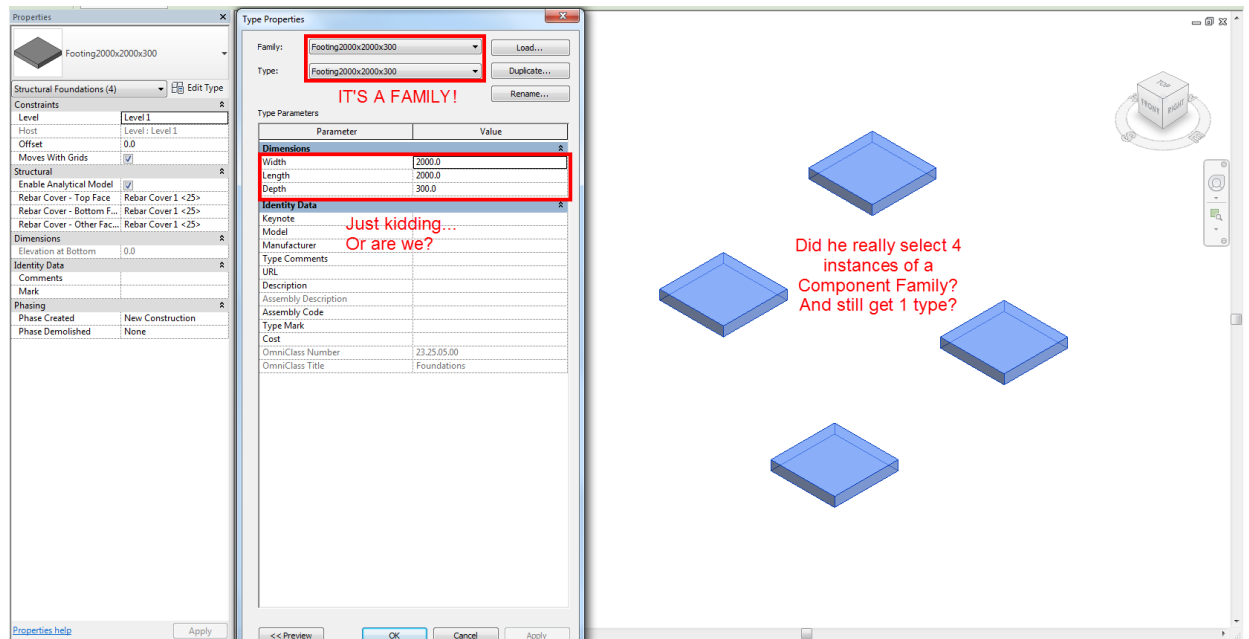


Image 31 this looks promising... but is it?

I'll leave the rest up to your imagination, and see you in Class...

In conclusion

Where is this all going to? What's the end goal? Why bother? Let me try to answer that with a few observations.

A few days ago, Jared Banks (aka @Shoegnome, www.shoegnome.com) wrote a post about the "I" in BIM. Here's a short fragment:

What if we've had it wrong this whole time? What if the I in BIM isn't Information. What if it's Intelligence. We went from pencil to CAD (Computer-Aided Design) to BIM (Building Information Models). But what if we are really just still in the process of moving to real BIM, to Building Intelligence Models.

In my very humble opinion he's not far from being completely right. In my opinion the general meaning of the word "information" is incomplete. Information, as a lot of people see it, is the same as data: if we put enough information in the model, then we're doing a good job. I beg to differ.

Information exists not just by sending it, but most of all because somebody (or something) receives it. If data isn't processed, it's not information. It's just 0's and 1's. Useless digital junk. Think about it: if I send a Facility Manager my rvt-file, while he/she has no access to Revit, nor the ability to process that rvt, am I actually providing him with information?

What if I send him an IFC derived from my model? Even though the actual data (properties) in that model are a fraction of the data present in the original rvt-file?

The IFC is accessible. Beauty of Open Source, it's always accessible. Tell me: when am I actually handing over information? Which one is the most valuable?

Interoperability to me is having the ability to send and receive data between different contributors of a project and while the receiving party actually has the ability to do something with it.

IFC is important because it **always** gives you the chance to process the data in the model, whether you need it tomorrow or in 50 years from now. No matter which software you use and if Autodesk even still exists. It might be vastly inferior (doubt that), but I don't care. Fact of the matter is: a lot of buildings I have modelled, even when they're just simple houses, will still be there after I'm dead. Those buildings will still be there when Revit is a thing from the far and distant past. In all those cases, the IFC scheme will also still be there. Giving anyone who wants to the ability to read the data inside my models and turn it into valuable, useable information.

As for the technical issues:

In this handout I've described workflows that are generally deemed impossible. IFC can't transfer information. IFC can't be parametric. Wrong. It can.

It can transfer every piece of data you want. Both out, and back in. It can be imported while remaining parametric constraints.

I know for a fact that it is indeed possible to update information when you receive a revised IFC file. It is possible to compare it to the one you already have, delete the items that are gone, update the objects that need updating and leave the rest untouched.

I know for a fact that it is possible to keep parametric constraints intact when importing an IFC.

I also know for a fact that it's possible to export any given piece of data to IFC, and to re-import it to the exact same place.

Put all of that together and the only thing standing between us and full interoperability is development. And that too is a beautiful thing about IFC: it's Open Source.

Wrap your head around this:

This is NOT Revit. This is NOT Tekla or Archicad. It's IFC. If you run into trouble on a project collaborating through IFC, you can fix it. You don't need to go to the Factory and ask if they can pretty please help you. There is no need for workarounds, there's no need to say "sorry, we can't do that".

In fact, chances are that if you did any moderately large or complex project that required working with IFC while still using the standard IFC tools in Revit, you probably spent more money on trying to work around a half-baked implementation than it would have cost you to hire an expert and make it work.

The stuff I showed you is developed by people who offered their spare time. Who, besides doing this project, had to worry about paying bills and feeding their kids. And deadlines... And who, in the process, achieved things that **none of the big "Open BIM" software vendors can do.**

Think about the possibilities when these people actually get paid for their work.