buildingSMART International
Modeling Support Group

IFC 2x Edition 3
Model Implementation Guide

Thomas Liebich

Version 2.0
May 18, 2009

Document Control

Editor	Thomas Liebich
Development Committee	Model Support Group
Project Reference	IFC Maintenance
Document Reference	IFC Implementation Guide
Document Version	Version 2.0
Release Date	May 18, 2009
Status	For Comments – please send to tl@aec3.com
Distribution	Public
Distribution format	PDF file

Acknowledgement

Major parts of the material for this documents had been developed by AEC3 Ltd. working at the Integrated Building Plan, Integrated Building Services (IBP/IBS) Plan Checking System carried out for the Building and Construction Authority of the Republic of Singapore (BCA) within the CORENET project. The IBP/IBS consortium is led by novaSPRINT Singapore and includes AEC3 Ltd. UK and EPM Norway.

BCA has kindly agreed to share and publish the document together with the International Alliance for Interoperability.

Table of Content

Table of Figures

Table of Tables

0 Document history

Version 2.0 beta – issues on 07.10.2009

The version 2.0b is the second pre-release leading towards version 2.0. The following additional changes and additions have been:

- general update of the overview section to reflect recent status of web resources
- intermediate publication, no update of other content yet

Version 2.0 alpha – not publicly issued

The version 2.0a is the first pre-release leading towards version 2.0. Version 2.0 has the goal to concentrate more on the implementation requirements and will be accompanied by an IFC Model Reference Guide. All more general explanations of the model have been moved over to the reference guide. As a consequence the implementation guide has been restructured.

The following additional changes and additions have been:

- addition of an introduction and a getting started section, see 1 and see 2
- addition of a section on the uppermost project container class, see 3.1
- addition of an appendix listing tools supporting IFC implementation, see Appendix 2

Version 1.8 – not publicly issued

The following changes have been made to the previous version of the document:

- addition of an appendix that include a complete list of all architectural entities for occurrences and related types, see Appendix 1.1
- new section about 2D representation and presentation added, see 9.2
- section on multiple geometric representation contexts has been enhanced, see 3.4
- section of geometric set representation has been enhanced by the IFC2x2 new subtype for geometric curve set, see 9.1.4.1.2

Version 1.7 – issued on 18.03.2004

The following changes have been made to the previous version of the document:

- addition of an appendix that include a complete list of all building service entities for occurrences and related types, see Appendix 1.2
- section on building service types enhanced, see 6.1.1
- update of section on element quantities for method of measurement, see 10.3.1.1
- section on assigning multiple classification notations corrected (wrong examples) see sections 10.4.2.2 and 10.4.2.7.1
- a note that requires style information to be always given to doors and windows has been added, see sections 5.4.1 and 5.4.2
- a sub section about multi-layered walls with different layer heights is added, see 5.2.2.4
- section about logical connectivity added, see 6.2.1
- section about specific guidelines to implement different building service elements added, see 6.3, first sub sections included are on flow moving devices 6.3.1, energy conversion devices 6.3.2, and on flow storage devices 6.3.3

Version 1.6 – issued on 30.06.2003

The following changes have been made to the previous version of the document:

- general update to IFC2x Edition 2, all sections have been reviewed to be compatible
- section on IFC2x Edition 2 (final and platform) delivery added (now part of reference guide)
- section of property values (bounded value 10.1.6.3 and list value 10.1.6.4) added and enhanced
- section on conversion based units corrected (wrong example about “Degree”), see 3.3.2.2
- section on specific walls to enforce a single body for shape representation, see 5.2.2
- section of slabs added, see 5.5
- section of openings in slabs added, see 5.3.4
- introduction into IFC2x2 concepts for building service elements, see 6.1
- introduction into IFC2x2 connectivity for building service elements, see 6.1.2.3
- introduction into IFC2x2 performance history for building service elements, see 6.1.2.3

Version 1.5 – issued on 23.04.2003

The following changes have been made to the previous version of the document:

- section on difference of IFC2x final and IFC2x platform added (now part of reference guide)
- section on “concept of multiple geometric contexts” enhanced and sub section 3.4.1 added
- section on polygonal walls added, see sub section 5.2.2.2 and 5.2.2.3
- section on special cases of wall to wall and wall to column connections, see sub section 5.2.3
- section on special opening type added, see sub section 5.3.3
- section of layer set directions enhanced for round walls, see 5.2.1.4
- section on using relative placement and elevation attribute for building story added, see 4.2.3.1

Version 1.4 – issued on 05.07.2002

The following changes have been made to the previous version of the document:

- section on openings added, including
  - openings in straight walls
  - openings in round walls
  - niches and recesses
- initial section on fillings added, including
  - doors with door style and predefined properties
  - windows with window style and predefined properties
- update of the section on building service systems

Version 1.3 – issued on 30.04.2002

The following changes have been made to the previous version of the document:

- section on spatial structures greatly enhanced, including
  - site, building, building story
  - space and space boundary
- section on building service elements added, including
  - geometric representation of building service elements
  - type / occurrence concept for building service elements
  - property set definitions on type and occurrence side

Version 1.2 – issued on 30.09.2001

The following changes have been made to the previous version of the document:

- update on geometric representation of products

  - shape aspects
  - clipping representation
- section on unique identification, state and ownership added
- section on quantities and measurements of elements added
- section on building service systems and zones added
- initial section on fundamentals for spatial structures added

Version 1.1 – issued on 10.08.2001

The following changes have been made to the previous version of the document:

- update on geometric representation of products
  - mapped representation
- section on fundamentals for building elements added, including
  - standard walls (straight and round walls)
  - special walls

Version 1.0 – issued on 29.06.2001

1 Introduction

This document targets the Construction IT community by providing guidelines on implementing the IFC standard. It is an extension to the existing IFC literature, particularly the IFC schema specification, with focus on implementation using STEP implementation technologies.

The IFC schema specification is distributed as the master EXPRESS definition, the HTML based online documentation of the EXPRESS schema, and an accompanying XML schema (XSD) definition, generated from the EXPRESS original. The specifics of the XML implementation using the XSD distribution form are discussed in a different document, the ifcXML implementation guide.

1.1 History

The IFC implementation guide has been first published following the introduction of IFC2x in year 2000. Since then, new versions of the IFC2x implementation guide have been regularly published with updates to the specific sections within the guide. In year 2003 the release IFC2x Edition 2 (short IFC2x2) had been introduced and it included support of different new domains. A new audience of application developers is expected and the IFC2x2 implementation guide should provide a reference also to those newcomers.

In December 2005 the next release had been published, IFC2x Edition 3 (short IFC2x3). The main focus of IFC2x3 has been a consolidation of the new scope as introduced in IFC2x2, including several bug-fixes, enhanced features, and better documentation.

In July 2007 a technical corrigendum has been published, the IFC2x3 TC1. It corrects some minor errors in the specification and improves the documentation. All differences to previous releases are noted in the “change log” and “what’s new” section of the IFC online specification.

1.2 Purpose

The IFC implementation guide is intended to provide a guidance for the first practical steps of implementing sections of the IFC schema, and as a reference to implementation agreements for commonly implemented parts of IFC.

It is not the purpose to provide complete instruction of an IFC implementation – these are guided by the IFC model view definitions.

Note: All implementations of IFC are implementations of a so called “IFC view”, or “Model View Definition”, examples in the implementation guide are based on the “coordination view”.

The implementation guide includes an introduction into using the IFC schema specification, the various additional documents needed for successful implementations, such as view definitions, the tools available to support IFC implementations, such as toolboxes and viewers, and examples of IFC files, containing different parts of an IFC building information model.

Note: The terms "schema" and "model" are used with the following meaning:

> *A **schema** is a collection of entities (or classes), attributes, and relationships between entities. It defines the patterns or templates by which populations of these entities and relationship shall be represented. Such a schema is often called a Product (Data) Model (as opposed to a populated data model). The IFC specification is a schema.*
>
> *A **model** is a population of a schema, following the patterns, templates and constraints stipulated by the schema. It contains the actual instances of the entities (or classes). Such a model is often called a populated data model, a project data model, a building information model (if content is construction industry specific. An IFC exchange file represents a building (information) model.*

1.3 Target audience

The target audience for this document is envisaged to include AEC-FM software application developers, who may be familiar with the IFC schema, but also developers and implementers from other domains.

1.4 Required skills

Readers should have a basic understanding of the IFC schema (best in its latest release IFC2x3) and access to the online IFC schema specification.

Developers should also have a basic background knowledge of EXPRESS based technology, including the EXPRESS data definition language (ISO10303-11:1994) and the STEP physical file format (ISO10303-21:1994). The implementation guide uses file examples written using the STEP physical file syntax.

1.5 Basic references

This guide should be read in close conjunction with the IFC schema documentation. Reference to the IFC schema specification is made as the ultimate authority. The IFC schema and accompanying documents, including this implementation guide, are published on the IAI International website with the URL:

- http://www.iai-tech.org

The official IFC EXPRESS specifications and the IFC model implementation guide can be accessed at:

- http://www.iai-tech.org/products/ifc_specification/ifc-releases/summary

The documentation of the latest IFC schema specification is available online at:

- for IFC2x3 TC1 (the most recent published release)
- http://www.iai-tech.org/products/ifc_specification/ifc-releases/ifc2x3-tc1-release/summary

1.5.1 Further reading

Although deep knowledge of EXPRESS and STEP physical file format is not required, a good understanding of how to read the IFC schema using EXPRESS definitions and the IFC file content in STEP physical file format in highly advisable.

A short overview on the EXPRESS language and the STEP physical file is available at:

- http://en.wikipedia.org/wiki/ISO_10303-11
- http://en.wikipedia.org/wiki/ISO_10303-21

The full specification is not available online, since the publication of ISO documents is protected by the ISO copyright that prohibits free online publications. If of interest, the complete specifications need to be purchased from the local ISO document publisher.

1.5.2 Newsgroups and other supporting resources

1.5.2.1 Implementation support group

IFC implementation is also guided by implementers agreements which are monitored by the buildingSMART Implementation Support Group (ISG), made up of active software developers, and assistance can be sought from them individually or collectively if necessary.

The home page of the ISG can be accessed at:

- http://www.iai.fhm.edu/

A listing of current implementer agreements can be accessed at:

- http://www.iai.hm.edu/how-to-implement-ifc/Implementer-Agreements

A listing of AEC/FM applications that have implemented IFC support is available at:

- http://www.buildingsmart.de/2/2_01_01.htm

ISG also maintain online collaboration space (using http://www.groove.net) and email exploders for officially registered buildingSMART members to support and coordinate implementation.

1.5.2.2 Model support group

IFC development, maintenance and improvement, as well as arbitrating implementation issues is carried out by the buildingSMART Modelling Support Group (MSG), made up of industry specialist in data modeling and implementing data models.

The home page of the ISG can be accessed at:

- http://www.iai-tech.org

Content, beside the published IFC schemas, include ifcXML, the property reference data, supporting material (including this implementation guide) and some source code and test cases.

Model view definitions providing additional guidance for implementing particular views (corresponding to particular exchange use cases) are developed and available under

- http://mvd.buildingsmart.com

1.5.2.3 Certification group

1.6 Document conventions

The following formatting conventions are used throughout the document:

- Extracts from IFC EXPRESS schemas and IFC data files are presented in Courier New
- Hyperlinks are colored blue and are underlined: http://www.iai-international.org/
- IFC entities are in italics and colored blue for example *IfcSpaceBoundary*
- IFC properties and attributes in running text are quoted for example ‘Longitude’

For compatibility with other IFC standard documents, spelling is American English.

2 Getting started

The IFC schema has as its scope to: "*to define a specification for sharing data throughout the project life-cycle, globally, across disciplines and across technical applications*". It is obvious from the scope statement that the IFC schema is inherently complex and broader than most of the actual exchange data needs, that are specific to one or a few disciplines and project life cycles. Therefore most practical IFC implementations are governed by a view definition.

- A "view" is a subset of the IFC schema that contains the data specification for a particular use case, or exchange scenario.

More information on currently available view definitions can be found at:

- {{Resource added later}}

2.1 Technologies used in IFC development

The IFC schema is specified using the EXPRESS data definition language, as defined in ISO10303-11:1994. EXPRESS data files have the file extension *.exp. Currently EXPRESS version 1 is used for defining the IFC schema.

IFC data files are clear text files following the STEP physical file format, as defined in ISO10303-21:1994. For public recognition the file extension for IFC data files is *.ifc (in contrary to standard STEP physical file extensions, being *.spf or *.p21). IFC files following the STEP physical file format are governed by the IFC schema in EXPRESS.

ISO10303-21 describes two conformance classes of implementation, one and two (or CC1 and CC2). For IFC implementations, conformance class one is used[1].

2.1.1 EXPRESS and XML Schema

Since Release IFC2x the IAI has published an XML schema definition file (file extension *.xsd) that equally defines the IFC schema. Here the primary EXPRESS data definitions have been converted into XSD using a language binding. Since Release IFC2x Edition 2 the language binding used is governed by ISO10303-28. Please refer to the ifcXML implementation guide for further reading.

- http://www.iai-international.org/Model/IFC(ifcXML)Specs.html

The figure 1 shows the correlation between the schema definition (the definition of classes (entities, elements) and types) that governs the population of model data files, and the exchange structure (documents, building models).

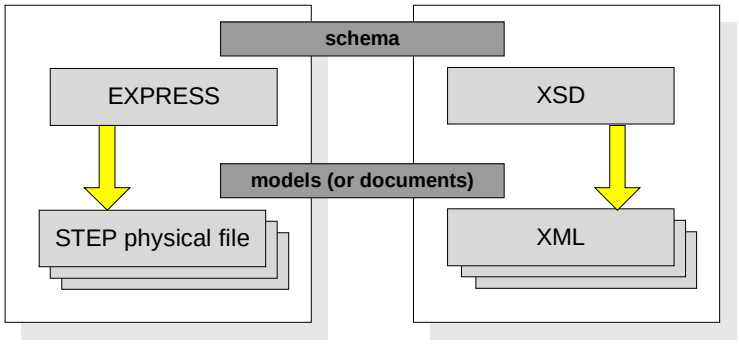


Figure 1 : Schema and model differentiation for EXPRESS and XML

[1] Conformance class one (CC1) uses internal mapping, i.e. all attributes (including those inherited from supertypes) are mapped into a single entity instance (within the sequence of its declaration from the supertype root to the leaf note subtype). Conformance class two (CC2) uses external mapping, i.e. all attributes are mapped into a complex entity instance, that is a combination of partial entity values, each refering to a specifc supertype of the entity instance. The latter is required to support non-exclusive subtype constraints (AND, ANDOR) that are available in EXPRESS, but not used in IFC. Refer to ISO10303-21:1994 for further reading.

2.1.2 STEP physical file and XML document

As shown in figure 1 an IFC model can be exchanged and shared using either the ISO 10303-21 “the STEP physical file” encoding or the XML document structure, the different forms are governed and can be checked against either the EXPRESS based, or the XML schema based IFC schema specification.

Currently the STEP physical file encoding is the preferred file structure to exchange IFC models.

- A short introduction into STEP physical file structure can be found here {{Resource added later}}
- Refer to ISO10303-21:2002 for further reading.

2.1.2.1 Structure of the STEP physical file

The “.ifc” file is a STEP physical file, SPF, and thereby a structured ASCII text file. The basic SPF structure divides each file into a header and a data section. The header section has information about:

- the IFC version used
- the application that exported the file
- the date and time when the export was done
- (often optionally) the name, company and authorizing person of the file

Example

```
HEADER;
FILE_DESCRIPTION(('IFC 2x platform'),'2;1');
FILE_NAME(
   'Example.dwg',
   '2005-09-02T14:48:42',
   ('The User'),
   ('The Company'),
   'The name and version of the IFC preprocessor',
   'The name of the originating software system',
   'The name of the authorizing person');
FILE_SCHEMA(('IFC2X2_FINAL'));
ENDSEC;
```

The data section contains all instances for the entities of the IFC specification. These instances have a unique (within the scope of a file) STEP Id, the entity type name and a list of explicit attribute.

Example

```
DATA;
#7=IFCCARTESIANPOINT((0.,0.,0.));
#8=IFCDIRECTION((0.,0.,1.));
#9=IFCDIRECTION((1.,0.,0.));
#10=IFCAXIS2PLACEMENT3D(#7,#8,#9);
ENDSEC;
```

The STEP Id is only valid for a single exchange, if the same user project is exported a second time, these Id’s will change. The order of the instances within the data section is not significant.

2.1.2.2 Encoding in STEP physical file structure

String data type

The characters allowed within a string without encoding are the decimal equivalents 32 through 126 (inclusive) of ISO 8859-1 that define the graphic characters of the basic alphabet (i.e. the ASCII code characters 32-126). All other characters have to be encoded.

“ISO 10303-11 defines the use of the control directive "\S\" to indicate a character in the basic alphabet to represent the character in the corresponding position in the extended alphabet. The control directive shall be interpreted in the string as the single character G((x+8)/y), where G(x/y) is the basic alphabet character following the "\S\". That is, if the basic alphabet character has code value v, it shall be interpreted as the character with code value v + 128.” [ISO 10303:2002 p.11]

Examples:

- a umlaut → \S\d → ä
- A umlaut → \S\D → Ä
- o umlaut → \S\v → ö
- O umlaut → \S\V → Ö
- u umlaut → \S\| → ü
- U umlaut → \S\\ → Ü

"An 8-bit byte with a value between 0 and 255 may be encoded in a string. The control directive "\X\" shall be used in a string to indicate that the following two hexadecimal characters shall be interpreted as an 8-bit byte representing the cell octet of a character in row 0 of the BMP in ISO 10646. The characters defined by ISO 10646 and ISO 8859-1 are identical within this range." [ISO 10303:2002 p.13]. It refers to the ASCII code characters 0-255, or hex 00-00FF.

- a umlaut → \X\E4 → ä
- A umlaut → \X\C4 → Ä
- o umlaut → \X\F6 → ö
- O umlaut → \X\D6 → Ö
- u umlaut → \X\FC → ü
- U umlaut → \X\DC → Ü
- a with small o umlaut → \X\E5 → å
- A with small o umlaut → \X\C5 → Å

The control directive "\X\" can also be used to encode non printable characters, e.g. new line.

- 'line one\X\0Aline two' line one
line two

ISO 10303-11 specifies that any character from ISO 10646 may occur in a string. It specifies three control directives that allow encoding of characters from ISO 10646. *"ISO 10646 defines a canonical form that uses four octets to represent any character in the full coding space. These characters specify the group, plane, row, and cell respectively. In addition, ISO 10646 defines a Basic Multilingual Plane (BMP), representing plane 00 of group 00 of the full coding space. The characters in the BMP are represented by two octets, specifying row and cell."* [ISO 10303:2002 p.12]

Note ISO/IEC 10646 and Unicode have been synchronized character for character and code for code. Any reference to ISO 10646 may be seen as a reference to UNICODE.

ISO 10303-11 defines the use of the control directive "\X2\" to indicate the following sequence of multiples of four hexadecimal characters shall be interpreted as encoding the two-octet representation of characters from the BMP in ISO 10646, and the use of the control directive "\X4\" to indicate the following sequence of multiples of eight hexadecimal characters shall be interpreted as encoding the four-octet representation of characters from the full coding space of ISO 10646. In either case, the control directive "\X0\" is used to indicate the end of encoding of ISO 10646 characters in a string and a return to direct encoding in the basic alphabet.

Note When using an EXPRESS based toolbox (see 2.2) functions are provided for basic text encoding following the ISO 10303-11 standard.

2.2 Tools to support the implementation of IFC

Most implementations of IFC support in AEC/FM applications are based on an **IFC toolbox**, a tool that provides the functionality to read and write IFC files in STEP physical file format and the API to map the AEC/FM applications instances into the IFC instances (and vice versa). The API usually provides one or more programming languages (like C++, Java, VB). Other functionality may include persistent database storage, support of ifcXML reading and writing, or others.

Other tools that are helpful for IFC development are:

- **IFC file validators**: tools that validate an IFC data file against the IFC schema, those tools can ensure, that the generated IFC data file is syntactically correct.
- **IFC geometry viewers**: tools that can read an IFC file and display the geometric items contained in the IFC file as 3D or 2D geometry. Often those viewers also allow to see the project structure

(building, stories, spaces contained in the IFC project data file), the components (listing of the walls, beams, etc. included) and properties of each component (the property sets, etc.)

- **IFC file browser**: tools that open an IFC file and allow to navigate along the instance references, those tools can be used to browse through the clear text file.

A listing of implementation support tools currently available is available at the IAI International website:

- http://www.iai-international.org/software/Tools%20for%20IFC%20developers.html

Please note that this listing is neither complete nor does it reflect any preference by the author in regard to the tools and the sequence in which they are included.

2.3 Releases of the IFC standard

3 Project Container Element and Base Settings

3.1 Project Container

Any IFC database and exchange file[2] requires the presence of exactly one instance of *IfcProject*.

Note: *There can only be a single instance of IfcProject within an IFC file or an IFC database model. The IfcProject instance holds the global definitions for the presentation context and the units within the global context – information which can only be provided once within the IFC file.*

The *IfcProject* establishes the context of all representation within the project. It includes:

- the default global units used within the IFC file or database model (see 3.3)
- the single world coordinate system, optionally referencing several scale and projection specific representation sub contexts (see 3.4), each defining:
  - the view and scale under which they apply
  - the coordinate space dimension
  - the precision used within the geometric representations, and
  - optionally the indication of the true north relative to the world coordinate system

In addition the attributes *Name*, *Description*, *ObjectType*, *LongName* and *Phase* allows to add further information to the *IfcProject* instance.

```
#1=IFCPROJECT('2mifZkraLA4xt6Lfl2GhCQ', #2, 'Shopping mall',
   'New development project in the suburban district', 'Development project',
   'Shopping mall at High Street', 'Design', (#3), #4);
```

The *IfcProject* is also the uppermost container class instance to which all products (as instances of *IfcProduct* or more specifically *IfcElement*) need to relate directly or indirectly. The highest spatial container class instance (often *IfcSite* or *IfcBuilding*) is directly assigned to the *IfcProject* using the *IfcRelAggregates* relationship.

3.2 Unique Identification, State and Ownership

The ability to uniquely identify an object as well as preserve information about its ownership is fundamental to the IFC model. These concepts are required for all subtypes of *IfcRoot* and are captured within the utility resource schema of the IFC2x specification.

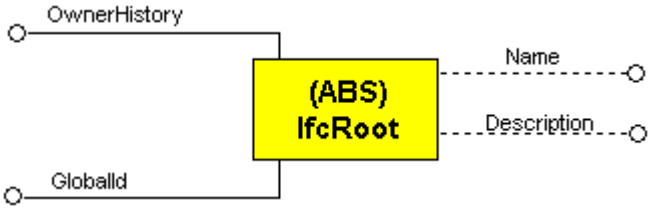


Figure 2 : Definition of identification and ownership at root object

3.2.1 Unique Identification of an Instance

Once an IFC instance is created, it is critical that it can be uniquely identified for its entire lifecycle. The ability to both exchange and archive data is not possible without this capability. For example, if multiple software applications are used to create instances of *IfcDistributionElement*, having a unique identifier associated with each instance allows for exchange and storage in a common archive without fear of namespace collisions.

The *IfcGloballyUniqueId* is an IFC defined type that is a fixed length string value. The stored value is a compressed Globally Unique Identifier (GUID) or Universal Unique Identifier (UUID) as defined by the Open Group. The IFC2x specifications contains a public domain algorithm that can be used to create,

[2] With the potential exception of message based IFC fragments within a model server environment, where the IFC data set does not need to be self sufficient.

compress and decompress these values. Every time an IFC instance is created, this algorithm must be used to create a new *IfcGloballyUniqueId*. Re-use of an *IfcGloballyUniqueId*, even if the local instance is deleted, is strongly discouraged due to the inability to ensure that remote instances do not exist.

3.2.2 State and Ownership of an Instance

IfcOwnerHistory is a lightweight definition that facilitates the ability to capture the following information:

- state information about the instance,
- the name and organization of the user that created and last modified the instance,
- the software application that created and last modified the instance,
- the date that the instance was created and last modified.

Information about the creating and last modified application is saved in instances of *IfcApplication*. Instances of both *IfcOwnerHistory* and *IfcApplication* can be shared.

Storing the state of an instance is optional, but highly recommended. State information is recorded in the *IfcOwnerHistory.State* attribute and allows applications to assert whether the instance should be modified by applications that are not the instance owner. Refer to the IFC2x specifications for details on available states.

Information about the user that created the instance is stored in the *IfcOwnerHistory.OwningUser* attribute. The intent is that the user that creates the instance is the owner. There is nothing that prevents an application from transferring ownership, and in some cases this may be necessary. For example, an architect may create instances of air terminals for the initial layout of a ceiling grid, but an HVAC engineer will ultimately need to take ownership of these instances.

If a user makes a change to an instance, information about the user is stored in the *IfcOwnerHistory.LastModifyingUser* attribute. Note that only the information about the *last* user is stored: a historical record is not preserved and is outside of the scope of this release of the IFC2x model. The nature of the change is recorded in the *IfcOwnerHistory.ChangeAction* attribute. Applications that do not own the instance should only allow changes to instances whose state is marked as READWRITE.

Application information consists of the name and version of the software application, the application developer, and the registered application identifier. The application identifier is granted by the IAI to the developer to indicate that the application has been approved by the IAI facilitated approval process.

Example:

The following Part 21 data contains an example of state and ownership concepts for an IfcBuilding instance that has just been created:

```
#1=IFCBUILDING('WJ7yGd42kMW08n7Cp432Kr',#2,'Sample Building',$,$,$,$,$,$,$,$);
#2=IFCOWNERHISTORY(#3,#6,.READWRITE.,.ADDED.,87763554,#3,#6,87763554);
#3=IFCPERSONANDORGANIZATION(#4,#5,());
#4=IFCPERSON('Public','Jane','Q.',(),(),(),(),());
#5=IFCORGANIZATION($,'Architecture by Jane Q. Public, Inc.',$,(),());
#6=IFCAPPLICATION(#7,'Version 1.0','Building Architecture Toolkit','BAT1.0');
#7=IFCORGANIZATION($,'Creating Instance Software, Inc.',$,(),());
```

3.3 Unit Definition and Assignment

In the IFC schema there may be a number of ways to express the measure defined types (for measure values) and their units. The purpose of this section and the examples is to promote common understanding how measure types and related units should be used, i.e. instantiated.

The definitions of units and measure types are provided in the *IfcMeasureResource* and had been based on the specification of the measure_schema as defined in ISO 10303-41 "Integrated Generic Resources – Fundamentals of Product Description and Support".

Note: An IFC project file always has units defined for its shape representations. There are no IFC project files without a unit definitions (in contrary DXF files do not have units).

3.3.1 Introduction

In different types of units there are four basic cases:

- Basic SI-units, which cover a number of fundamental units of mainly physical quantities defined by ISO-1000+A1,1992 & 1998.; e.g. meter or millimeter as unit for length measure or square meter as a unit for area measure. The unit may have a scaling prefix (milli, kilo, etc.).
- Conversion based units, which can be derived (by direct scaling) from SI-units; e.g. inch which can be defined using SI-units for length measure, i.e. an inch is 25.4 millimeters.
- Derived units, which can be defined as a derivation or combination of a number of basics units together with their dimensional exponent in that unit, e.g. kg / m2.
- Context dependent units, which cannot be directly defined as conversion based unit using SI-units.

With regard to the usage of measure defined types (*IfcLengthMeasure*, *IfcTimeMeasure*, etc.) as attribute data types in the IFC schema, there two basic cases:

```
ENTITY IfcCartesianPoint
   SUBTYPE OF (   IfcPoint);
      Coordinates     :    LIST [1:3] OF IfcLengthMeasure;
   ...
END_ENTITY;
```

- The data type of an entity attribute is a measure defined type as such. In this case it's the global unit assignment for the corresponding unit for this measure type that defines the unit for all the usages of this defined measure type. This is the case for all geometry definitions (there is only a single length unit for all geometric items within an IFC file) and for most of the other properties.

```
ENTITY IfcPropertySingleValue
   SUBTYPE OF (   IfcSimpleProperty);
      NominalValue   : IfcValue;
      Unit           : OPTIONAL IfcUnit;
END_ENTITY;
```

- For some generic property definitions, e.g. within the general property set or quantity take-off definitions, there is the ability to define local units. The convention is that if a specific unit is given, than this unit overrides a potentially provided global unit for the same measure, if no local unit is given, the globally defined unit is used. The data type of an unit is *IfcUnit*, which allows for definition of an unit as either SI unit, derived unit, conversion based unit or context dependent unit (or any valid combination).

In general, it is recommended not to mix different units for same measure defined types, if it can be avoided. The next subsections give some examples of each of the above cases.

> Note*: In the example instantiations in the form of IFC data exchange files, only the measure and unit -relevant attributes are given the values; the other attributes are given no values (in the form of $-sign) independent of whether they should actually have values because of being non-optional attributes.*

3.3.2 Global unit assignment

The global unit assignment of the project defines the global units for measures and values when the units are not otherwise defined more specifically using entity type *IfcMeasureWithUnit* as attribute's data type. The global unit assignment is handled at the *IfcProject* (see 3.1).

The relevant entity within the *IfcMeasureResource*, which is used to list all globally defined units, is the *IfcUnitAssignment*.

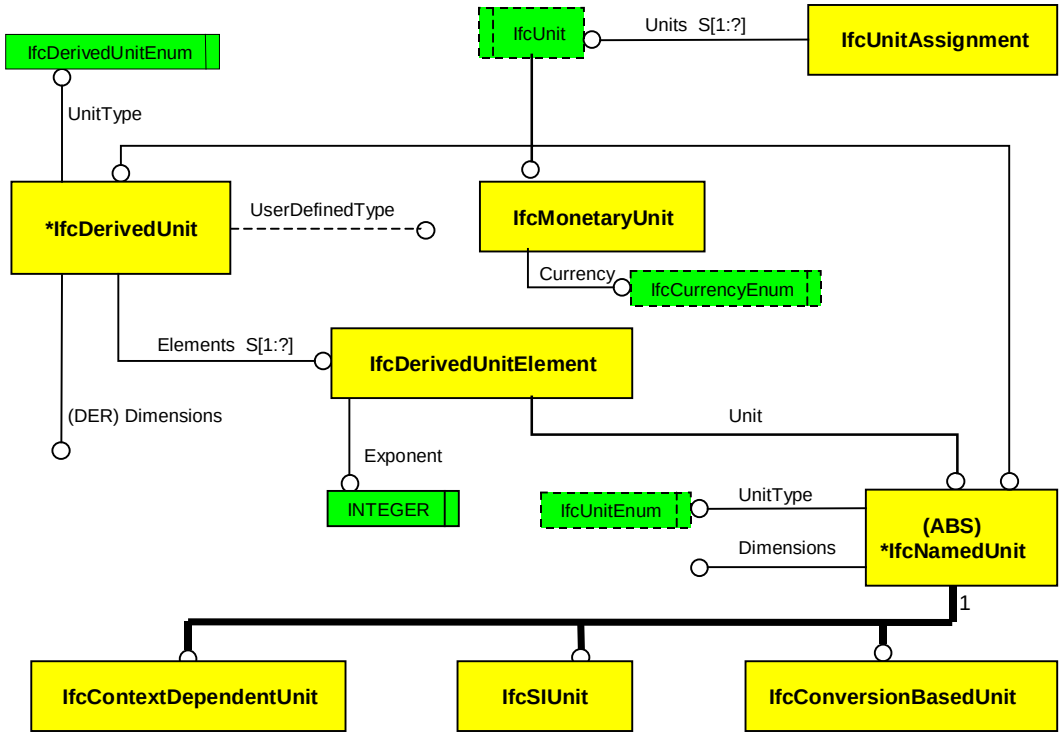


Figure 3 : Definition of IfcUnitAssignment and the IfcUnit select

3.3.2.1 Basic SI-units as global units

SI units are defined by the SI unit name, provided as an enumeration of all SI units, as defined in ISO 1000. A Prefix (such as milli, kilo, tera, etc.) can be provided in addition. If it is omitted, the basic SI unit is given.

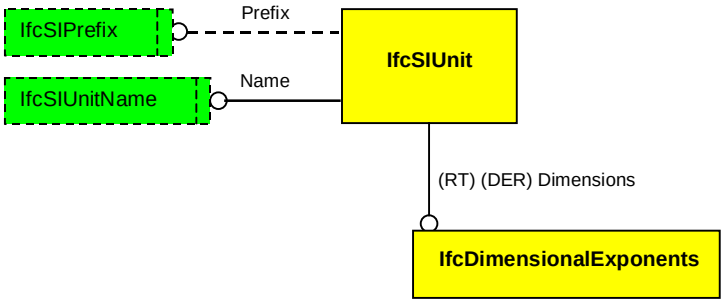


Figure 4 : Definition of IfcSIUnit

An example where project's global basic length, area and volume units are defined as SI units. Here all length measures are given as millimeter, all area measures as square meter and all volume measures as cubic meter.

```
#1= IFCPROJECT('fabcdeghijklmnopqrst02',#7,'test project',$,$,$,$,(#20),#30);
/* global units */
#30=IFCUNITASSIGNMENT((#33, #34, #35));
#33=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
```

All measures within the geometric representations of IFC objects (see chapter 9) are given by measures with defined data types referring back to the global units, i.e. all lengths are given by the same unit (e.g. millimeter as in the example above).

Another example is the exchange of the global unit for angle measures (e.g. the plane angle measures for rotations). Here the unit is given by radian.

```
#1= IFCPROJECT('fabcdeghijklmnopqrst02',#7,'test project',$,$,$,$,(#20),#30);
#30=IFCUNITASSIGNMENT((#33, #34, #35, #36));
#33=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#36=IFCSIUNIT(*. .PLANEANGLEUNIT., $, .RADIAN.);
```

3.3.2.2 Conversion based units as global units

Global units within a project could also be given as derived units (although the use of standard SI units is encouraged). In this case the *IfcUnitAssignment* should refer to the derived unit definition, given by the unit type, a name and the conversion rate in regard to the SI units for the same unit type.

The *ValueComponent* of the *IfcMeasureWithUnit* that is used as *ConversionFactor* should given the amount of the base units, that equals 1 conversion based unit.

Examples

- *IfcConversionBasedUnit* = “Inch” – *ValueComponent* of *ConversionFactor* = 25.40005
- *IfcConversionBasedUnit* = “Degree” – *ValueComponent* of *ConversionFactor* = 0.0174532926

If degrees should be used for plane angle measures, in contrary to the example above (i.e. 180’ instead of 3.1416, or π), than it has to be declared as a derived unit, referring to the radian as the underlying SI unit.

```
#1= IFCPROJECT('fabcdeghijklmnopqrst02',#7,'test project',$,$,$,$,(#20),#30);
#30= IFCUNITASSIGNMENT((#33, #34, #35, #36));
#33= IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34= IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35= IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#36= IFCCONVERSIONBASEDUNIT(#40, .PLANEANGLEUNIT., 'DEGREE', #41);
#40= IFCDIMENSIONALEXPONENTS(0, 0, 0, 0, 0, 0, 0);
#41= IFCMEASUREWITHUNIT(IFCPLANEANGLEMEASURE(57.29577951308232), #50);
#41= IFCMEASUREWITHUNIT(IFCPLANEANGLEMEASURE( 0.01745329263638), #50);
#50= IFCSIUNIT(*. .PLANEANGLEUNIT., $, .RADIAN.);
```

Implementation agreements: The conversion factor (in the above example #41) should store the value the sending system had used for the generated IFC file. The receiving system can use that value, if it is however in contradiction to the internally preferred conversion factor, the system has the freedom to choose its internal value.

3.3.2.2.1 Conversion based Imperial units as global units

An example where projects global basic length and area units are defined as imperial units (inches and square feet), which are further defined as conversion based units relative to SI units millimeter and square meter. In the example SI unit cubic meter is defined as the global volume unit:

```
#1=IFCPROJECT($, $, '', $, $, $, $, $, #2, $, $);
#2=IFCUNITASSIGNMENT((#6, #9, #5));
#3=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#4=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#5=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#6=IFCCONVERSIONBASEDUNIT(#8, .LENGTHUNIT., 'INCH', #7);
#7=IFCMEASUREWITHUNIT(IFCLENGTHMEASURE(25.40005), #3);
#8=IFCDIMENSIONALEXPONENTS(1, 0, 0, 0, 0, 0, 0);
#9=IFCCONVERSIONBASEDUNIT(#10, .AREAUNIT., 'SQUARE_FEET', #11);
#10=IFCDIMENSIONALEXPONENTS(2, 0, 0, 0, 0, 0, 0);
#11=IFCMEASUREWITHUNIT(IFCAREAMEASURE(0.0929), #4);
```

3.3.2.3 Derived units as global units

An example definition of a unit for specific heat capacity (Joule / kg Kelvin), which is defined as a derived unit based on basic SI units:

```
#1=IFCPROJECT('abcdefghijklmnopqrst09', #100, '', $, $, $, $, (#101), #2);
#2=IFCUNITASSIGNMENT((#3));
#3=IFCDERIVEDUNIT((#5, #6, #4), .SPECIFICHEATCAPACITYUNIT., $);
#4=IFCDERIVEDUNITELEMENT(#7, 1);
#5=IFCDERIVEDUNITELEMENT(#8, -1);
#6=IFCDERIVEDUNITELEMENT(#9, -1);
#7=IFCSIUNIT(*, .ENERGYUNIT., $, .JOULE.);
#8=IFCSIUNIT(*, .MASSUNIT., .KILO., .GRAM.);
#9=IFCSIUNIT(*, .THERMODYNAMICTEMPERATUREUNIT., $, .KELVIN.);
```

3.3.2.4 Use of defined measure type with global unit assignment

An example where the attribute *IfcScheduletimeControl.ScheduleDuration* : *IfcTimeMeasure*, is of data type *IfcTimeMeasure* without possibility of using specific unit from *IfcMeasureWithUnit*. Then the global unit for time measure applies; in this case seconds (the example's 172800 seconds would be equivalent to 2 days):

```
#1=IFCPROJECT('abcdefghijklmnopqrst09', #100, '', $, $, $, $, (#101), #2);
#2=IFCUNITASSIGNMENT((#3));
#3=IFCSIUNIT(*, .TIMEUNIT., $, .SECOND.);
#4=IFCSCHEDULETIMECONTROL($, $, $, $, $, $, $, $, $, $, $, $, $, 172800., $, $, $, $,
   $, $, $);
```

3.3.3 Local unit assignment

An example where the *IfcRelAssignsToProcess.QuantityInProcess* : *IfcMeasureWithUnit* of data type *IfcMeasureWithUnit* is given a value in liters (value 10 liters e.g. as a quantity value for paint) as a conversion based unit based on SI unit cubic meter. Here an "overriding" specific unit can be defined on the instance level independent of the global unit for volume measure (here SI unit cubic meter):

```
#1=IFCPROJECT('abcdefghijklmnopqrst10', #100, '', $, $, $, $, $, #2, $, $);
#2=IFCUNITASSIGNMENT((#8));
#3=IFCRELASSIGNSTOPROCESS'abcdefghijklmnopqrst11', #100, $, $, (#51,#52), $, #53, #4);
#4=IFCMEASUREWITHUNIT(10., #5);
#5=IFCCONVERSIONBASEDUNIT(#6, .VOLUMEUNIT., 'LITRE', #7);
#6=IFCDIMENSIONALEXPONENTS(3, 0, 0, 0, 0, 0, 0);
#7=IFCMEASUREWITHUNIT(IFCVOLUMEMEASURE(0.001), #8);
#8=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
```

3.4 Representation context

Each *IfcRepresentation* (and the subtype *IfcShapeRepresentation*) has to be founded within a representation context, established by the *IfcProject* (see 3.1), referencing a single or multiple instances of *IfcRepresentationContext* (or the subtype *IfcGeometricRepresentationContext*).

An IFC project has to have at least one representation context, but may have several contexts. Each context maybe identified by two attributes, *ContextIdentifier* and *ContextType*. Currently there are no recommendations on how to use the *ContextIdentifier* attribute, but there are recommendations on how to use the *ContextType*. Additional recommendations maybe be given within particular views or implementation agreements.

Within IFC2x2 a new subtype of *IfcGeometricRepresentationContext* has been added, the *IfcGeometricRepresentationSubContext,* it refers to its parent context and derives the *WorldCoordinateSystem*, the *CoordinateSpaceDimension*, the *TrueNorth* and the *Precision*. The intended usage of the *IfcGeometricRepresentationSubContext* is the handling of multi-view representation of the same object within the same representation context (i.e. the same coordinate space). The two attributes *TargetScale* and *TargetView* allow to specify the representation set, to which a specific shape or styled representation can be assigned to.

Example: The representation of a table might be different within the general model space (3D) and a specific representation for the 1:100 ground view. It can be achieved in IFC2x2 by assigning two IfcRepresentation instances (e.g. by the IfcShapeRepresentation and IfcStyledRepresentation subtypes) within the IfcProductDefinitionShape for the IfcFurnitureElement, and to assign the IfcStyledRepresentation (1:100 ground view) to an IfcGeometricRepresentationSubContext with TargetScale=0.01 and TargetView=.PLAN_VIEW.

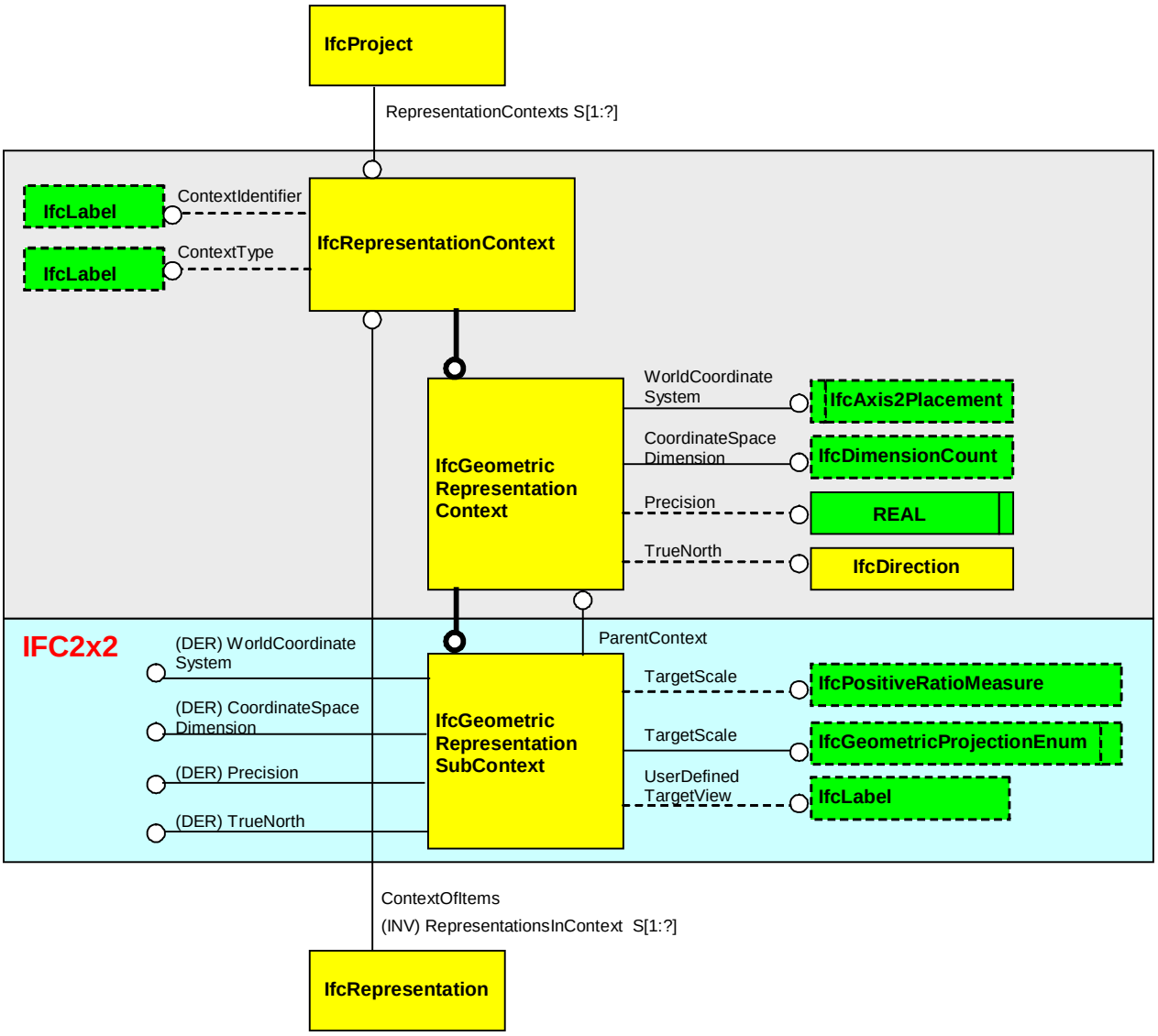


Figure 5 : Definition of (multiple) representation contexts

In case of geometric representations of a product definition shape (as described in 9.1.3), the subtype *IfcGeometricRepresentationContext* has to be used (since IFC2x2 also the new subtype of *IfcGeometricRepresentationContext*, the *IfcGeometricRepresentationSubContext* can be used to differentiate between several scale and/or view dependent representations within the same geometric context). In addition to the *ContextIdentifier* and the *ContextType*, the subtype provides two mandatory attributes:

- *WorldCoordinateSystem* : Establishment of the world coordinate system for the representation context used by the project. It is used directly for the UoF "local absolute placement" (see 9.1.2.1.2), where the object coordinate system in directly placed within the WCS, as established by the geometric representation context. In case of the UoF "local relative placement" (see 9.1.2.1.1), the uppermost relative placement must relate to the WCS, as established by the geometric representation context3.
- *CoordinateSpaceDimension* : The integer dimension count of the coordinate space used in the geometric representation context.

The two optional attributes *Precision* and *TrueNorth* may give additional information about the context.

[3] If there are multiple geometric representation contexts within a project, all are required to refer to the same world co-ordinate system. In case of 2D and 3D geometric co-ordination contexts, the XY-plane has to be identical.

- *Precision* : Value of the model precision for geometric models. It is a double value (REAL), typically in 1E-5 to 1E-8 range, that indicates the tolerance under which two given points are still assumed to be identical. The value can be used e.g. to sets the maximum distance from an edge curve to the underlying face surface in brep models. This information is needed for the reliable exchange of B-rep shape models, and should be provided, if shape representations of representation type "Brep" are used.
- *TrueNorth* : Direction of the true north relative to the world coordinate system as established by the representation context. This information maybe provided for the benefit of thermal analysis applications.

There is a constraint, that all instances of *IfcGeometricRepresentationContext* have to define the same world coordinate system, if there are both, 2D and 3D geometric contexts, that they have to define identical XY planes.

The new IFC2x2 subtype *IfcGeometricRepresentationSubContext* has an *ParentContext* attribute, that refers to the *IfcGeometricRepresentationContext* and establishes the same *WorldCoordinateSystem*, *CoordinateSpaceDimension*, *Precision,* and *TrueNorth* for all sub contexts of the same *ParentContext*. This enforces, that all geometric representation sub contexts to have an identical world coordinate system and thereby the same position and distances between its coordinates.

3.4.1 Context type of geometric representation context

If there are multiple geometric representation contexts used the attributes the *ContextIdentifier* and *ContextType* should be used to distinguish the contexts. This functionality allows to support the concept of multi-view representations (such as multi-view blocks or different display representation of the same object depending on scale and projection). Current recommendations for the label of *ContextType* are:

- graph view,
- sketch view,
- model view,
- plan view,
- reflected plan view,
- section view,
- elevation view

Example:

All bounding box shape representations of the objects should be grouped within a separate IfcGeometricRepresentationContext with the ContextType = “sketch view”.

Note:

A new subtype, IfcGeometricRepresentationSubContext, has been defined in IFC2x2, which provides additional capabilities for the definition of such context type specific sub contexts, it has two explicit attributes for defining the TargetView and the TargetScale, at which the sub context applies.

4 Spatial structure and space elements

All spatial structure elements inherit the concepts associated to products, which are provided by *IfcProduct*. It includes:

- "Identification" and "Change Management", see section 3.1.
- "Generic Type Assignment" and "Generic Property Attachment", see section 9.2.
- "External References" to documents, classifications and libraries, see section 10.4.
- "Generic Decomposition" redefined as decomposition of the spatial structure within this section.
- "Placement" and "Shape Representation", see section 9, refined as spatial structure element specific shape representation (also referred to as geometry use cases) within this section.
- "Element in Space Containment", introduced within this section.
- "Element Quantities", see section 10.3.

4.1 Common concepts for all spatial structure elements

The spatial structure can be defined as "*breakdown of the project model into manageable subsets according to spatial arrangements*". It should be noted, that other decomposition structures for a project exist, but that the spatial structure is common to most disciplines and design tasks. It is therefore seen as the primary structure for building projects and required for the data exchange.

The following four different concepts are subsumed under the *IfcSpatialStructureElement* entity:

- Project (uppermost container of all information)
- Site (also site complex and part of site)
- Building (also building complex and building section)
- Building story (also partial building story)
- Space (also partial space)

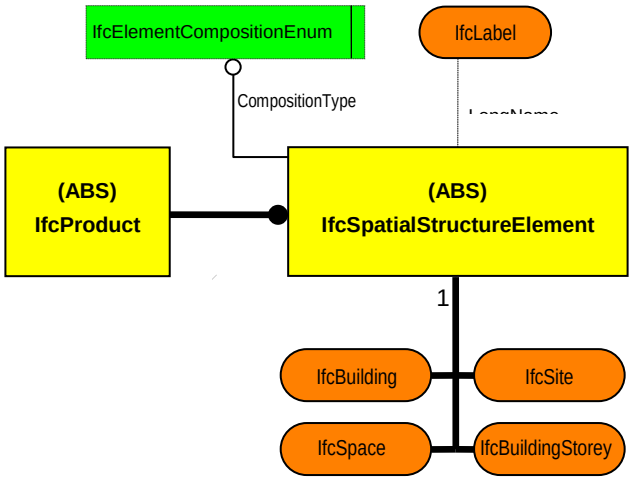


Figure 6 : Definition of spatial structure elements

These different entities are contained by each other such as they provide a clear hierarchical structure for the building project. The next figure shows such a structure.

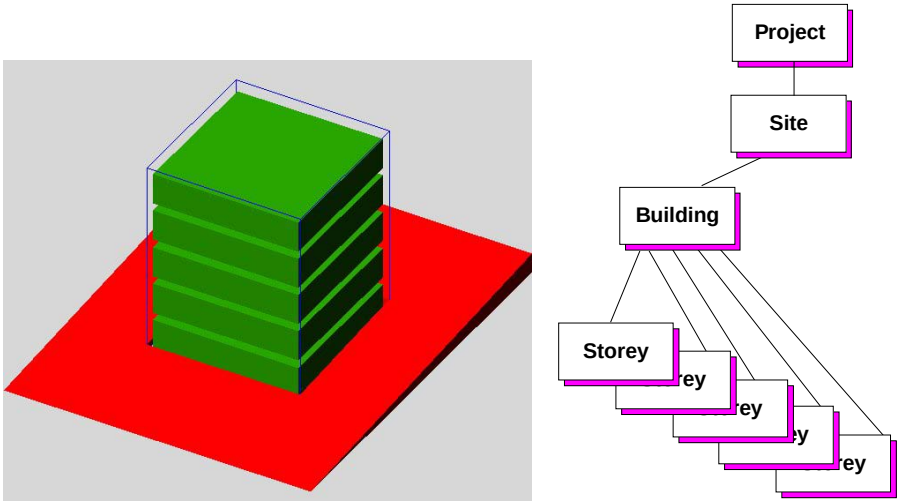


Figure 7 : Spatial structure of a building project

The four subtypes *IfcSite, IfcBuilding, IfcBuildingStorey, IfcSpace* are used to represent the levels of the spatial structure. Each has

- a name (by *IfcRoot.Name*)
- a description (by *IfcRoot.Description*)
- a long name

The *Name* attribute has to be used to store the name of the building, and building story. E.g., the story name, such as “First Floor” should be exchanged by *Name*.

```
#9=IFCBUILDINGSTOREY('abcdefghijklmnopqrs109', #109, 'First Floor', $, $, $, $, $,
.ELEMENT., $);
```

The various subtypes define further attributes, which are introduced in the appropriate sub sections. In total each of the subtypes inherit the following attributes and inverse relationships:

```
ENTITY IfcSpatialStructureElement
   GlobalId         : IfcGloballyUniqueId;
   OwnerHistory     : IfcOwnerHistory;
   Name             : OPTIONAL IfcLabel;
   Description      : OPTIONAL IfcText;
   ObjectType       : OPTIONAL IfcLabel;
   ObjectPlacement  : OPTIONAL IfcObjectPlacement;
   Representation   : OPTIONAL IfcProductRepresentation;
   LongName         : OPTIONAL IfcIdentifier;
   CompositionType  : IfcElementCompositionEnum;
INVERSE
   IsDefinedBy      : SET OF IfcRelDefines;
   HasAssociations  : SET OF IfcRelAssociates;
   HasAssignments   : SET OF IfcRelAssigns;
   Decomposes       : SET OF IfcRelDecomposes;
   IsDecomposedBy   : SET [0:1] OF IfcRelDecomposes;
   ReferencedBy     : SET OF IfcRelAssignsToProduct;
   ContainsElements : SET OF IfcRelContainedInSpatialStructure;
END_ENTITY;
```

4.2 Building the spatial structure

The spatial structure of a building project is created by using the fundamental decomposition relationship. The subtype *IfcRelAggregates* is used to link the instances of *IfcProject*, *IfcSite, IfcBuilding, IfcBuildingStorey, IfcSpace* (provided that all levels are applicable) and establishes an hierarchical structure.

usual building structure

(1...1) IfcProject
(0...n) IfcSite
(1...n) IfcBuilding (total)
(0...n) IfcBuilding (section)
(1...n) IfcBuildingStorey (total)
(0...n) IfcBuildingStorey (partial)

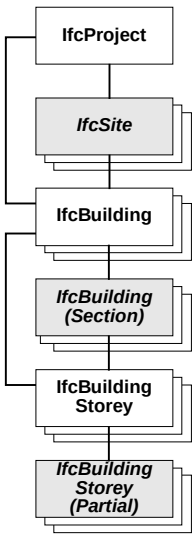


Figure 8 : Mandatory and optional levels of the spatial structure

Whereas the *IfcProject, IfcBuilding* and *IfcBuildingStorey* are mandatory levels for the exchange of complex project data, the *IfcSite* and *IfcSpace* represent optional levels (which may be provided, if they contain necessary data, but don't need to be provided).

In order to define a complex or a partial spatial structure element, the *CompositionType* attribute is used. If the *CompositionType* is COMPLEX, then it defines a site complex, or a building complex. If the *CompositionType* is PARTIAL, then it defines a partial site or a building section. If the *CompositionType* is ELEMENT, it defines (just) the site or the building.

Each instance of *IfcProject*, *IfcSite, IfcBuilding, IfcBuildingStorey, IfcSpace* is connected to other instances of the spatial structure by an instance of *IfcRelAggregates,* where the single *RelatingElement* points to the element at the higher level and the 1 to many *RelatedElements* point to the elements at the lower level of the hierarchy.

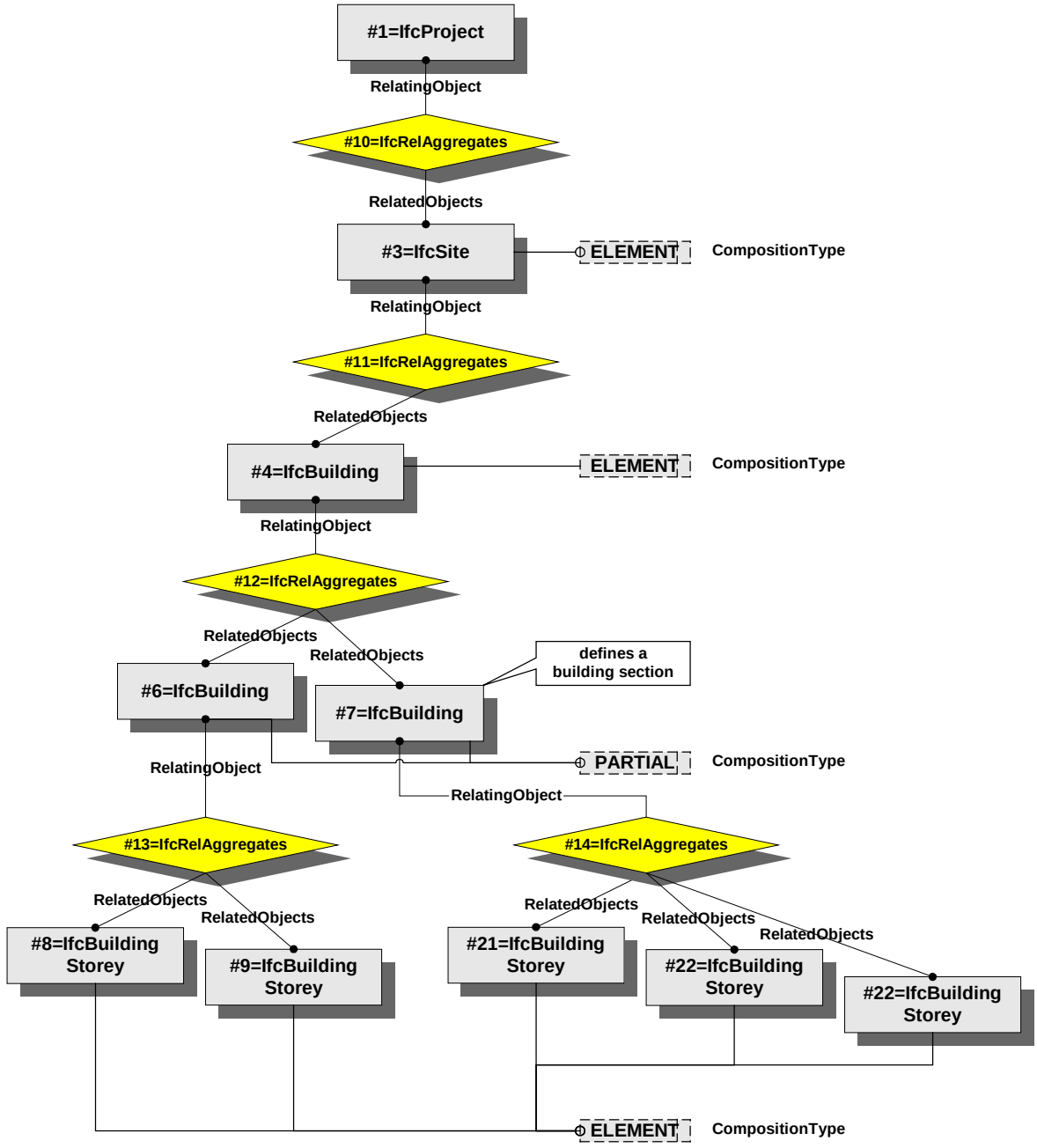


Figure 9 : Decomposition of a spatial structure

Note: The Figure 8 shows the use of the IfcRelAggregates to define a spatial structure of a building project having a single site with one building. The building is further divided into two building section. Two stories are assigned to the first building section and three stories are assigned to the second building section.

The Figure 8 shows a vertical and horizontal division of the building. In these cases, the horizontal division (into building sections) takes priority and the building stories are later assigned to the sections. A building story that physically spans through two building sections would therefore be subdivided into two building stories, and each then assigned to a single building section. Figure 9 shows a graphical image of the sample spatial structure.

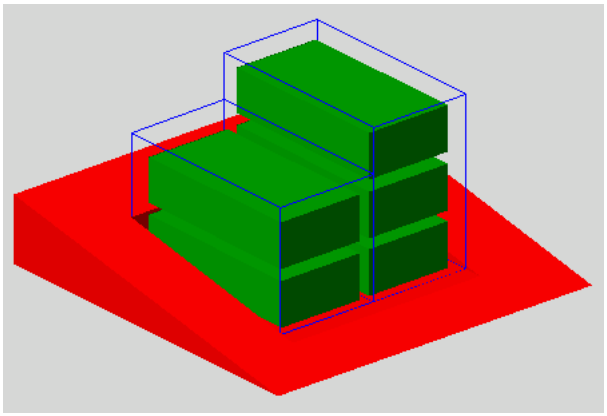


Figure 10 : Layout of the example spatial structure

The example file shows the basic spatial structure of the example shown in Figure 8 and Figure 9. It should be noted, that the following information has to be given for each project:

- the *Name* (enforced by where rule: WR1)
- at least one *RepresentationContexts*, being an *IfcRepresentationContext* (or subtype)
- the *UnitsInContext* to define the global units

```
#1=IFCPROJECT('abcdefghijklmnopqrs101', #101, 'sample project', $, $, $, $, (#1000),
   #1010);
#3=IFCSITE('abcdefghijklmnopqrs103', #103, $, $, $, $, $, $, .ELEMENT., $, $, $, $, $);
#4=IFCBUILDING('abcdefghijklmnopqrs104', #104, $, $, $, $, $, $, .ELEMENT., $, $, $);
#6=IFCBUILDING('abcdefghijklmnopqrs106', #106, $, $, $, $, $, $, .PARTIAL., $, $, $);
#7=IFCBUILDING('abcdefghijklmnopqrs107', #107, $, $, $, $, $, $, .PARTIAL., $, $, $);
#8=IFCBUILDINGSTOREY('abcdefghijklmnopqrs108', #108, $, $, $, $, $, $, .ELEMENT., $);
#9=IFCBUILDINGSTOREY('abcdefghijklmnopqrs109', #109, $, $, $, $, $, $, .ELEMENT., $);
#21=IFCBUILDINGSTOREY('abcdefghijklmnopqrs121', #121, $, $, $, $, $, $, .ELEMENT., $);
#22=IFCBUILDINGSTOREY('abcdefghijklmnopqrs122', #122, $, $, $, $, $, $, .ELEMENT., $);
#23=IFCBUILDINGSTOREY('abcdefghijklmnopqrs123', #123, $, $, $, $, $, $, .ELEMENT., $);
#10=IFCRELAGGREGATES('abcdefghijklmnopqrs110', #110, $, $, #1, (#3));
#11=IFCRELAGGREGATES('abcdefghijklmnopqrs111', #111, $, $, #3, (#4));
#12=IFCRELAGGREGATES('abcdefghijklmnopqrs112', #112, $, $, #4, (#6, #7));
#13=IFCRELAGGREGATES('abcdefghijklmnopqrs113', #113, $, $, #6, (#8, #9));
#14=IFCRELAGGREGATES('abcdefghijklmnopqrs114', #114, $, $, #7, (#21, #22, #23));
```

4.2.1 Site

The site, represented by *IfcSite*, is used to provide additional information about the building site and may include the representation of the terrain model for the building site. The following additional information (to the inherited *Name, Description, LongName* and *ObjectType*) may be given for an *IfcSite*.

- the *RefLatitude, RefLongitude, RefElevation* to provide the world latitude, longitude and datum elevation of a reference point within the site. That information is given for information purposes and may be used within thermal or other applications,
- the *LandTitleNumber* which is used by many constituencies to identify the site,
- the *SiteAddress*, which is for information purposes only, but may be exchanged together with the virtual building model

In addition to the alphanumerical attributes the site mainly handles the reference to the building(s) which is(are) built or maintained at that site and the geometry of the terrain. Three different shape representations can be exchanged for the *IfcSite*:

1. the set of survey points and optionally breaklines, this is normally the outcome from the survey at the site and similar to existing simple exchange formats for site or terrain geometry. In this case the triangulation of the resulting face or shell based geometry (or mesh) is left to the receiving application.
2. the mesh of the terrain model, a set of 3 or 4 side faces connected in an connected face set, in this case the triangulation is already provided by the sending application

3. the b-rep geometry of the resulting solid (requires a fully closed solid for the terrain)

In case 1 the following applies to the *IfcShapeRepresentation* used (see Figure 10):

- the *RepresentationIdentifier* should be “SurveyPoints”;
- the *RepresentationType* “GeometricSet”;
- the *Items* should include an *IfcGeometricSet* containing all survey points as 3D *IfcCartesianPoint* and the breaklines as *IfcPolyline* connecting the selected survey points.

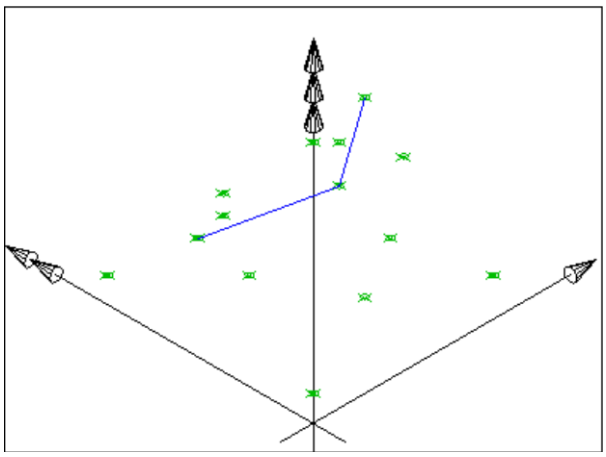


Figure 11 : site representation by survey points and breaklines

In case 2 the following applies to the *IfcShapeRepresentation* used (see Figure 11):

- the *RepresentationIdentifier* should be “Mesh”;
- the *RepresentationType* “SurfaceModel'”;
- the *Items* should include an *IfcFaceBasedSurfaceModel* containing one or many *IfcConnectedFaceSet* with the triangulated (or quadrilateral and co-planer) faces as *IfcFace*.

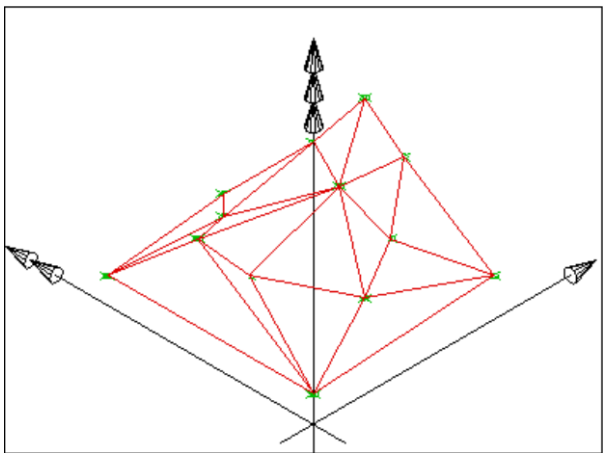


Figure 12 : site representation by a mesh

In case 3 the following applies to the *IfcShapeRepresentation* used:

- the *RepresentationIdentifier* should be “Body”;
- the *RepresentationType* “Brep”;
- the *Items* should include an *IfcFacetedBrep* or an *IfcFacetedBrepWithVoids* (following the general rules for B-rep Geometry (see 9.1.4.1.4.1).

4.2.2 Building

The building, represented by *IfcBuilding*, is used to provide additional information about the building itself. It is associated (if given) to an *IfcSite*, if not, than directly to the *IfcProject*.

A building includes the references to the building stories which belong to that building, or a building may have building sections, which would be referenced by the building, in that case the building stories are referenced by the building sections[4].

The following additional information (in addition to the inherited *Name, Description, LongName* and *ObjectType*) may be given for an *IfcBuilding*.

- the *ElevationOfRefHeight* which is needed to put the elevation datum of the stories into the global context of the elevation above sea level. The *ElevationOfRefHeight* gives the height relative to ± 0.00 (normally the elevation of the ground floor). Although not strictly required this information should be exchange if specified by the user in the source application.
- the *ElevationOfTerrain* gives the minimum elevation of the terrain at the foot print of the building, this is for information purposes only and may not be sufficient when the building is situated in a sloped terrain. In that case only the terrain model, associated to *IfcSite*, can provide this information geometrically.
- the *BuildingAddress* which is for information purposes only, but may be exchanged together with the virtual building model. It may, or may not, be identical with *IfcSite.SiteAddress*.

Usually the building does not contain its own shape representations, as it is provided by its constituting elements. The *ObjectType* attribute should be reserved to contain a simple type description of the building (following local building standards), if a more complex classification of the building is required, the proper classification resource should be used (see 10.4.2).

The example below describes a sample building structure with building specific information (name, type, long name and address given). Also note that unit assignment has mm as length unit (not shown), therefore elevation is given in mm. PS: No position is given in the example.

```
#1=IFCPROJECT('abcdefghijklmnopqrs101', #101, 'sample project', $, $, $, $, (#1000),
   #1010);
#3=IFCSITE('abcdefghijklmnopqrs103', #103, $, $, $, #1500, $, $, .ELEMENT., $, $, $, $,
   $);
#4=IFCBUILDING('abcdefghijklmnopqrs104', #104, 'sample building', $, 'office', #1501, $,
   'sample building at 100 main road', .ELEMENT., 129350., 128750., #1020);
#8=IFCBUILDINGSTOREY('abcdefghijklmnopqrs108', #108, $, $, $, $, $, $, .ELEMENT., $);
#9=IFCBUILDINGSTOREY('abcdefghijklmnopqrs109', #109, $, $, $, $, $, $, .ELEMENT., $);
#10=IFCRELAGGREGATES('abcdefghijklmnopqrs110', #110, $, $, #1, (#3));
#11=IFCRELAGGREGATES('abcdefghijklmnopqrs111', #111, $, $, #3, (#4));
#13=IFCRELAGGREGATES('abcdefghijklmnopqrs113', #113, $, $, #4, (#8, #9));

#1020=IFCPOSTALADDRESS(.SITE., $, $, $, ('100 Main Street'), $, 'Major Town',
   'Best Region', '123456', $);
```

4.2.3 Building Story

The building story, represented by *IfcBuildingStorey*, is used to provide the information about the building story itself, a vertical structure normally used within building and construction. It is always associated to an *IfcBuilding*, either the building or the building section.

A building story includes the references to the spaces which belong to that building story. A building story may have partial building stories (such as split levels), which would be referenced by the building story, in that case the spaces are referenced by the partial building stories[5].

The following additional information (in addition to the inherited *Name, Description, LongName* and *ObjectType*) may be given for an *IfcBuildingStorey*.

- the *Elevation* which is needed to identify the datum of the story (based on the raw construction of the slab), although the attribute is optional, it is necessary for many exchange scenarios (or views)

[4] It should be noted that the creation and exchange of such complex spatial structures (such as building sections) may be beyond current capabilities of software systems used in building/construction.

[5] It should be noted that the creation and exchange of such complex spatial structures(such as split level stories within normal stories) may be beyond current capabilities of software systems used in building/construction.

Usually the building story does not contain its own shape representations, as it is provided by its constituting elements. The *ObjectType* attribute should be reserved to contain a simple type description of the building story.

Such a description should be agreed on in a project or local context and may include identifiers as "Basement", "GroundFloor", "UpperFloor", or "RoofTop".

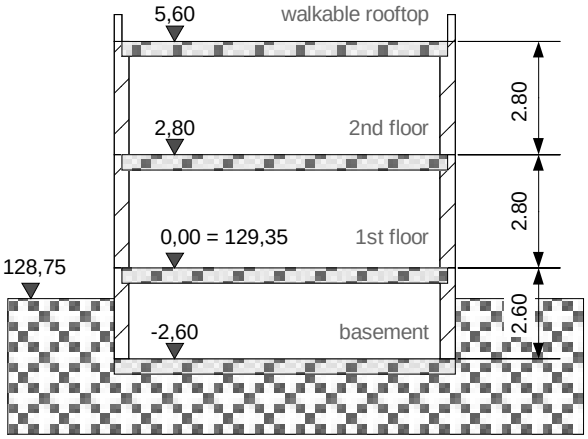


Figure 13 : Example of building story structure

The example below describes the building structure, as given in Figure 12, including all relative placements (see 9.1.2.1.1) and attributes as discussed.

```
#1=IFCPROJECT('abcdefghijklmnopqrs101', #101, 'sample project', $, $, $, $, (#1000),
   #1010);
#3=IFCSITE('abcdefghijklmnopqrs103', #103, $, $, $, #1500, $, $, .ELEMENT., $, $, $, $,
   $);
#4=IFCBUILDING('abcdefghijklmnopqrs104', #104, 'sample building', $, 'office', #1501, $,
   'sample building at 100 main road', .ELEMENT., 129350., 128750., #1020);
#6=IFCBUILDINGSTOREY('abcdefghijklmnopqrs106', #106, 'basement story', $, 'Basement',
   #1502, $, $, .ELEMENT., -2600.);
#7=IFCBUILDINGSTOREY('abcdefghijklmnopqrs107', #107, '1st story', $, 'GroundFloor',
   #1503, $, $, .ELEMENT., 0.);
#8=IFCBUILDINGSTOREY('abcdefghijklmnopqrs108', #108, '2nd story', $, 'UpperFloor',
   #1504, $, $, .ELEMENT., 2800.);
#9=IFCBUILDINGSTOREY('abcdefghijklmnopqrs109', #109, 'walkable rooftop', $, 'RoofTop',
   #1505, $, $, .ELEMENT., 5600.);
#10=IFCRELAGGREGATES('abcdefghijklmnopqrs110', #110, $, $, #1, (#3));
#11=IFCRELAGGREGATES('abcdefghijklmnopqrs111', #111, $, $, #3, (#4));
#13=IFCRELAGGREGATES('abcdefghijklmnopqrs113', #113, $, $, #4, (#6, #7, #8, #9));

#1500=IFCLOCALPLACEMENT($,#1600);
#1501=IFCLOCALPLACEMENT(#1500,#1601);
#1502=IFCLOCALPLACEMENT(#1501,#1602);
#1503=IFCLOCALPLACEMENT(#1501,#1603);
#1504=IFCLOCALPLACEMENT(#1501,#1604);
#1505=IFCLOCALPLACEMENT(#1501,#1605);

#1600=IFCAXIS2PLACEMENT3D(#1700,$,$);
#1601=IFCAXIS2PLACEMENT3D(#1701,$,$);
#1602=IFCAXIS2PLACEMENT3D(#1702,$,$);
#1603=IFCAXIS2PLACEMENT3D(#1703,$,$);
#1604=IFCAXIS2PLACEMENT3D(#1704,$,$);
#1605=IFCAXIS2PLACEMENT3D(#1705,$,$);

#1700=IFCCARTESIANPOINT((0.,0.,0.));
#1701=IFCCARTESIANPOINT((0.,0.,0.));
#1702=IFCCARTESIANPOINT((0.,0.,-2600.));
#1703=IFCCARTESIANPOINT((0.,0.,0.));
#1704=IFCCARTESIANPOINT((0.,0.,2800.));
#1705=IFCCARTESIANPOINT((0.,0.,5600.));
```

Beside spaces normally all building elements are assigned to the building story, in which they are located. If building elements (or spaces) span through many stories, they are assigned to the story at which they are based. Also elements that are linked to other elements, such as openings or doors and windows are separately assigned directly to the building story.

This containment relationship is handled by the UoF "Element in Space Containment". The relationship *IfcRelContainsInSpatialStructure* should be used to exchange it.

The spatial structure is also reflected in the relative positioning. Each story, building and site should declare its own object coordinate system, which is positioned relative to the coordinate system of the spatial structure at the next higher level (i.e. identical with the *RelatingObject* in the *IfcRelAggregates* containment relationship)[6].

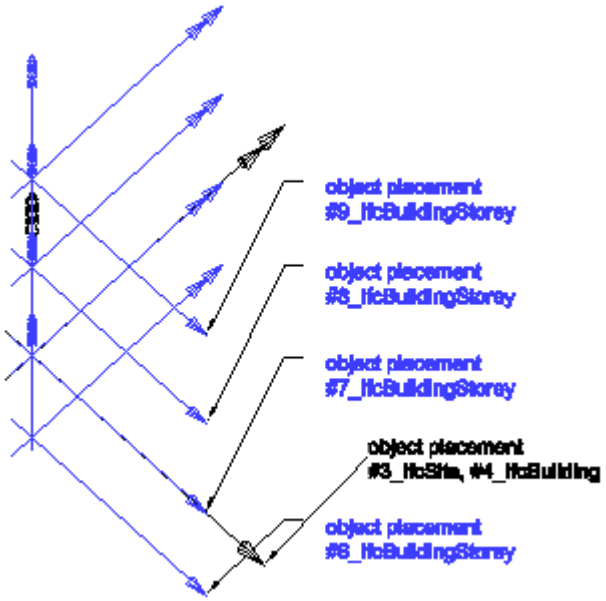


Figure 14 : relative object placement of the spatial structure

4.2.3.1 Story elevation

There are two scenarios for exporting multi-story buildings using the relative object placement and the *Elevation* attribute. These apply particularly to the coordination view of IFC2x.

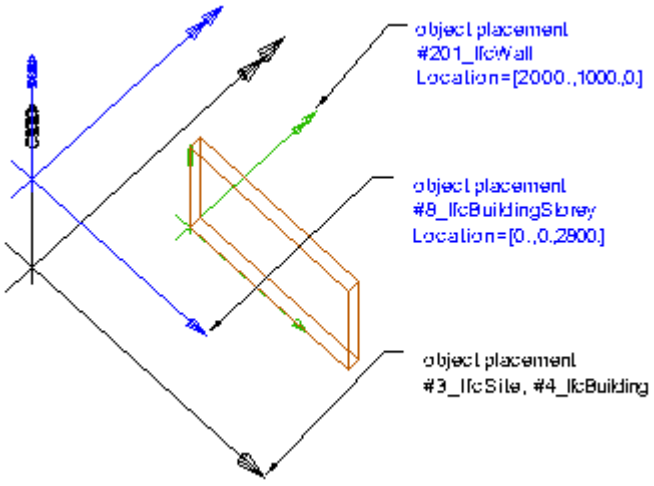


Figure 15 : Story elevation with relative placement

- the default scenario (to be used whenever possible) requires the use of relative object placements (as shown in Figure 14) and the use of the *Elevation* attribute.

[6] The relative placement can be omitted if only absolute placements (see 10.1.2.1.2) are used.

- in this case the first story of the previous example (#8) needs to have a relative local placement (usually relative to *IfcBuilding*) with the z-coordinate of the placement location being equal to the story elevation (here “2800.”) **and** the *Elevation* attribute set to the elevation (again “2800.”).
- all building elements, that are assigned to the building story are place relative to the building story, i.e. most walls would usually have a local placement with the z-coordinate of the placement location being “0.00”.

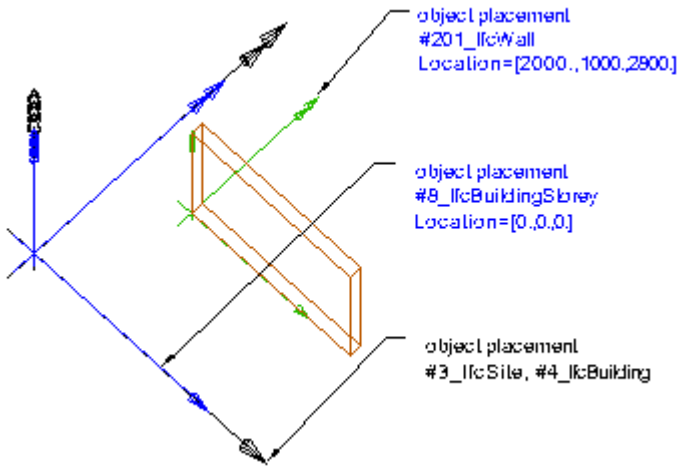


Figure 16 : Story elevation with placement height zero and elevation

- the fallback scenario (to be used when it is the user intention not to place stories on top of each other) uses of relative object placements without using the z-coordinate of the placement location but requires the use of the *Elevation* attribute, see Figure 15.
  - in this case the first story of the previous example (#8) is allowed to have a relative local placement (usually relative to *IfcBuilding*) with the z-coordinate of the placement location being equal to the building (here “0.”) **but** the *Elevation* attribute has to be set to the elevation (here “2800.”).
  - all building elements, that are assigned to the building story are place relative to the building story, but as it is set to equal height to the building, most walls would now usually have a local placement with the z-coordinate of the placement location being “2800.”.

4.2.4 Space

The space, represented by *IfcSpace*, is used to provide all information about the space as a functional area or volume within a spatial structure. It is associated (if given) to an *IfcBuildingStorey*, in the special case of an outdoor space, however, it can be directly associate to an *IfcSite*.

The following guidelines should apply for using the *Name, Description, LongName* and *ObjectType* attributes.

- the *Name* holds the unique name (or space number) from the plan.
- the *Description* holds any additional information field the user may have specified, there are no further recommendations.
- the *LongName* holds the full name of the space, it is often used in addition to the *Name*, if a number is assigned to the room, than the descriptive name is exchanged as *LongName*.
- the *ObjectType* holds the space type, i.e. usually the functional category of the space[7].

[7] This can only be a rough indication of the type, for proper classification according to a building code the light weight classification (see 11.4.2.7.1) should be used.

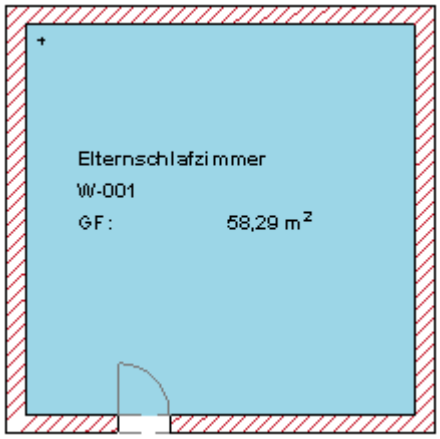


Figure 17 : room stamp with space information

The following partial file contains the snip-out of the space object with all name and simple type information, here in a German context[8].

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.0);
```

The following additional information (in addition to the inherited *Name, Description, LongName* and *ObjectType*) may be given for an *IfcSpace*.

- the *InteriorOrExteriorSpace* which is an enumeration, indicating whether the space is an internal space (i.e. fully enclosed by a building structure) or an external space,
- the *ElevationWithFlooring* which is needed to identify the datum of the finish of the space (based on the flooring on top of the raw construction of the slab).

Spaces normally contain all building service or interior design elements (such as distribution elements, electrical elements, furniture, fixture and equipment), if they are located within a single space. Large distribution elements, such as ducts or pipes, which span through multiple spaces can only be contained by the building story. This containment relationship is handled by the UoF "Element in Space Containment". The relationship *IfcRelContainsInSpatialStructure* should be used to exchange it.

If the classification of the space should be exchanged explicitly it should follow the UoF “external classification association”, particularly the “light weight classification” (see 10.4.2.7.1). Taking the previous example, the space is classified according to the German DIN277 as a HNF1. It should be exchanged as follows:

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.0);
#301=IFCCLASSIFICATIONREFERENCE($, 'HNF1', 'Wohnen und Aufenthalt', #302);
#302=IFCCLASSIFICATION('Deutsche Industrienorm', 'Juni 1987', $, 'DIN277');
#303=IFCRELASSOCIATESCLASSIFICATION('2hJ0MTr$v9hgPt_3AR_yCZ', #6, $, $, (#285), #301);
```

If the quantities should be exchanged explicitly it should follow the UoF “"Element Quantities" (see 10.3). Quantities of spaces normally include net or gross areas, volumes and perimeters. They are normally calculated according to a Method of Measurement, which has to be given as well in order to interpret the results.

The following Figure 17 shows the quantities of the example space. They are all measured according to the Method of Measurement DIN277.

[8] English translation of the example: type is living area and long name is master’s bedroom. Elevation with flooring is 0.0, i.e. the finish of the ground level.

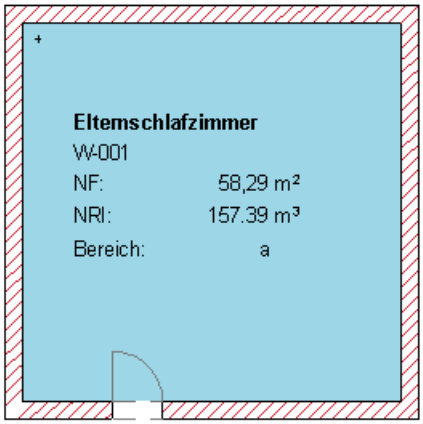


Figure 18 : Quantities of sample space

The exchange of the calculated values for area and volume should be done by using the *IfcElementQuantity* holding the Method of Measurement and the individual subtypes of *IfcPhysicalQuantity* for each value. Since the *Unit* attribute is NIL, the globally defined units apply to the values of the quantity measures.

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.);

#304=IFCELEMENTQUANTITY('2hJ0MTr$v9hgPt_3AR_yDZ', #6, 'Space Quantities', $, 'DIN277',
     (#306,#305));
#305=IFCQUANTITYAREA('HNF1', 'Net area', $, 58.29);
#306=IFCQUANTITYVOLUME('NRI a', 'Net volume', $, 157.39);
#307=IFCRELDEFINESBYPROPERTIES('2hJ0MTr$v9hgPt_3AR_yEZ', #6, $, $, (#285), #304);

#7=IFCSIUNIT(*, .LENGTHUNIT., $, .METRE.);
#8=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#9=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#10=IFCSIUNIT(*, .MASSUNIT., $, .GRAM.);
#11=IFCSIUNIT(*, .TIMEUNIT., $, .SECOND.);
#12=IFCSIUNIT(*, .THERMODYNAMICTEMPERATUREUNIT., $, .DEGREE_CELSIUS.);
#13=IFCSIUNIT(*, .LUMINOUSINTENSITYUNIT., $, .LUMEN.);
#14=IFCUNITASSIGNMENT((#7,#8,#9,#10,#11,#12,#13));
```

In addition to the alphanumerical attributes the space mainly holds the room geometry. Three different shape representations can be exchanged for the *IfcSpace*:

1. the footprint of the space – a 2D closed polyline or composite curve.
2. the 3D shape of the shape – a 3D body defined by extruding the footprint and potentially clipping the result.
3. the b-rep geometry to capture any 3D shape of spaces.

In case 1 the following applies to the *IfcShapeRepresentation* used:

- the *RepresentationIdentifier* should be 'FootPrint';
- the *RepresentationType* should be 'Curve2D', or 'GeometricCurveSet';
- the *Items* should include an *IfcCurve*, normally *IfcCompositeCurve* for any curve, or *IfcPolyline* for curves with only line segments.

Within the simple space example used in Figure 18 the shape presentation of the footprint is given by a polyline, as no arc segments are used.

NOTE: If a space has inner boundaries (i.e. subtraction areas) the RepresentationType should be 'GeometricCurveSet', allowing a set of Items to store the outer and inner boundaries.

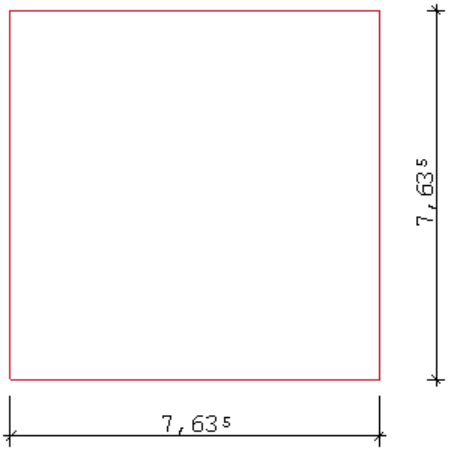


Figure 19 : footprint geometry of the sample space

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.);
#300=IFCPRODUCTDEFINITIONSHAPE($, $, (#400));
#400=IFCSHAPEREPRESENTATION(#50, 'FootPrint', 'Curve2D', (#401));
#401=IFCPOLYLINE((#402,#403,#404,#405,#402));
#402=IFCCARTESIANPOINT((0.0,0.0));
#403=IFCCARTESIANPOINT((7.635,0.0));
#404=IFCCARTESIANPOINT((7.635,7.635));
#405=IFCCARTESIANPOINT((0.0,7.635));
```

In case 2 the following applies to the *IfcShapeRepresentation* used:

- the *RepresentationIdentifier* should be 'Body';
- the *RepresentationType* should be 'SweptSolid' (or 'Clipping');
- the *Items* should include an *IfcExtrudedAreaSolid*, normally having the *SweptArea* being an *IfcArbitraryClosedProfileDef*.

Within the simple space example used in Figure 18 (having a height of 2.7m) the shape presentation of the body is given by an extruded solid based on an arbitrary profile without voids.

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.);
#300=IFCPRODUCTDEFINITIONSHAPE($, $, (#500));

#500=IFCSHAPEREPRESENTATION (#70, 'Body', 'SweptSolid', (#501));
#501=IFCEXTRUDEDAREASOLID (#503, #502, #509, 2.7);
#502=IFCAXIS2PLACEMENT3D (#510, $, $);
#503=IFCARBITRARYCLOSEDPROFILEDEF (.AREA., $, #504);
#504=IFCPOLYLINE ((#505, #506, #507, #508, #505));
#505=IFCCARTESIANPOINT ((0., 0.));
#506=IFCCARTESIANPOINT ((7.635, 0.));
#507=IFCCARTESIANPOINT ((7.635, -7.635));
#508=IFCCARTESIANPOINT ((0., -7.635));
#509=IFCDIRECTION ((0., 0., 1.));
#510=IFCCARTESIANPOINT ((0., 0., 0.));
```

In case 3 the following applies to the *IfcShapeRepresentation* used:

- the *RepresentationIdentifier* should be 'Body';
- the *RepresentationType* should be 'Brep';
- the *Items* should include an *IfcFacetedBrep*, (with or without voids).

4.2.4.1 Space boundaries

The space boundaries define the relationship between a space and its bounding elements. Each space boundary defines an 1:1 relationship between a space and a single bounding building element (if given), in this case a physical space boundary. In cases of an open space, the open side is defined as a virtual space boundary.

The provision of space boundaries for spaces is declared in the UoF “Space Element Boundaries”. Space boundaries can be defined:

- logically, i.e. without a geometric representation of the boundary,
- physically, i.e. with a geometric representation given by a surface definition.

If space boundaries are defined physically then the geometry is given within the object coordinate system of the space[9]. For space boundaries defined along walls, the following geometry types should be used:

- for straight walls – *IfcSurfaceOfLinearExtrusion*, based on an *IfcArbitraryOpenProfileDef* with an *IfcPolyline* based *Curve*. (also *IfcTrimmedCurve* based on an *IfcLine* segment is possible).
- for curved walls – *IfcSurfaceOfLinearExtrusion*, based on an *IfcArbitraryOpenProfileDef* with an *IfcTrimmedCurve* based *Curve*.

In the simple sample space, a space boundary is defined as shown in Figure 19.

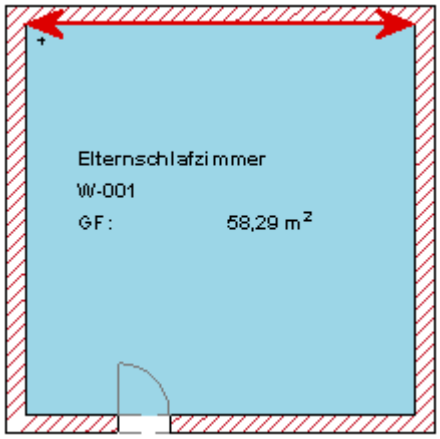


Figure 20 : space boundary of sample space

```
#285=IFCSPACE('3LweZaMsz0nR$8x1w5Bjie', #6, 'W-001', $, 'Wohnen und Aufenthalt', #284,
     #300, 'Elternschlafzimmer', .ELEMENT., .INTERNAL., 0.);
#1000=IFCWALLSTANDARDCASE('2yOJsOFC58PATD3uOMT1eh', #6, $, $, $, #1035, $, $);

#1036=IFCRELSPACEBOUNDARY('3LweZaMsz0nR$8x1w5Bklm', #6, $, $, #285, #1000, #1037,
     .PHYSICAL., .INTERNAL.);
#1037=IFCCONNECTIONSURFACEGEOMETRY(#1038, $);
#1038=IFCSURFACEOFLINEAREXTRUSION(#1039, #1040, #1041, 2.7);
#1039=IFCARBITRARYOPENPROFILEDEF(.CURVE., $, #1042);
#1040=IFCAXIS2PLACEMENT3D(#1045, $, $);
#1041=IFCDIRECTION((0.,0.,1.));
#1042=IFCPOLYLINE((#1043,#1044));
#1043=IFCCARTESIANPOINT((7.635,7.635));
#1044=IFCCARTESIANPOINT((0.,7.635));
#1045=IFCCARTESIANPOINT((0.,0.,0.));
```

4.2.5 Concept of Zone

Within the IFC Model, a zone is defined as a subtype of *IfcGroup*. That is, it acts as a functional related aggregation of objects. However, whilst the *IfcGroup* can aggregate any set of instances that are subtypes of *IfcObject*, the *IfcZone* is restricted by a WHERE rule so that it can only aggregate spaces or other zones.

```
ENTITY IfcZone
   SUBTYPE OF (IfcGroup);
   WHERE
      WR1 : SIZEOF (QUERY (temp <* SELF\IfcGroup.IsGroupedBy.RelatedObjects |
            NOT(('IFCPRODUCTEXTENSION.IFCZONE' IN TYPEOF(temp)) OR
                ('IFCPRODUCTEXTENSION.IFCSPACE' IN TYPEOF(temp))) )) = 0;
```

[9] The provision of the space boundary is not required in all views declared for the use of the IFC2x model. However some views, like the HVAC or code-checking view depend on the provision of space boundaries.

```
END_ENTITY;
```

The purpose of a zone is to be able to define spaces that are considered together for the purposes of some external requirement. That is, spaces may be aggregated into a zone for the purposes of provision of engineering services (the maximum load on the south side of a building occurring at a different time of day to the maximum load on the north side of the building), for fire compartmentation, sprinkler zoning, security zoning, office tenancy etc.

Whilst it is normally expected that the spaces in a zone will be contiguous (i.e. adjacent to each other), this is not a mandatory requirement. Therefore, it is possible to define a zone that includes non-adjoining spaces. These spaces may also be on separate storeys within the building. This is relevant for certain types of zoning that may only be applicable to particular types of space.

4.2.5.1 Zone Naming

There may be many instances of *IfcZone* within a building, each zone fulfilling a different purpose. To distinguish between them, it is important that each zone is named. Zone names should be defined using the inherited *Name* attribute from *IfcRoot*. This resolves to a simple STRING data type. Note also that the attribute *ObjectType* inherited from *IfcObject* may be optionally used to qualify the zone name further. This is particularly used in conjunction with property sets or type being attached to the zone.

5 Building Elements

All building element inherit the concepts and UoF associated to elements, which are provided by *IfcElement*. It includes:

- "Identification" and "Change management", see section 3.1.
- "Generic Type Assignment" and "Generic Property Attachment", see section 9.2, refined as element type assignment for selected building elements.
- "External References" to documents, classifications and libraries, see section 10.4.
- "Generic Decomposition" refined as decomposition of the building element structure within this section.
- "Placement" see section 9.1.2, and "Shape Representation", see section 9.1.3, refined as building element specific shape representation (also referred to as geometry use cases) within this section.
- "Element in Spatial Structure Containment", see section 3.
- "Element quantities", see section 10.3.
- "Material assignment", see section 10.2. refined as building element specific material assignment (e.g. for walls) within this section.
- "Element Connectivity", to be added later.

5.1 Common concepts for all building elements

All building elements are exchanged by an IFC2x file as instances of subtypes of the abstract entity *IfcBuildingElement*. If the appropriate semantic classification exists in the IFC2x specification (and within the originating application), than the building element is exported by the appropriate subtype entity, or (if this is not the case) by the *IfcBuildingElementProxy* entity.

Within Figure 20 all currently defined building elements are shown, those on top of the gray area are part of the stable IFC2x platform, those on white background are part of the additional domain layer.

The various subtypes define further attributes, which are introduced in the appropriate sub sections. In total each of the subtypes inherit the following attributes and inverse relationships from *IfcElement*: Elements also have the following additional attributes for the UoF "Space Element Boundaries" (see 4.2.4.1), "Element Opening Voiding" (see 5.3) and "Opening Element Filling".

```
ENTITY IfcElement
   GlobalId         : IfcGloballyUniqueId;
   OwnerHistory     : IfcOwnerHistory;
   Name             : OPTIONAL IfcLabel;
   Description      : OPTIONAL IfcText;
   ObjectType       : OPTIONAL IfcLabel;
   ObjectPlacement  : OPTIONAL IfcObjectPlacement;
   Representation   : OPTIONAL IfcProductRepresentation;
   Tag              : OPTIONAL IfcIdentifier;
INVERSE
   IsDefinedBy      : SET OF IfcRelDefines;
   HasAssociations  : SET OF IfcRelAssociates;
   HasAssignments   : SET OF IfcRelAssigns;
   Decomposes       : SET [0:1] OF IfcRelDecomposes;
   IsDecomposedBy   : SET OF IfcRelDecomposes;
   ReferencedBy     : SET OF IfcRelAssignsToProduct;
   ConnectedTo      : SET OF IfcRelConnectsElements;
   ConnectedFrom    : SET OF IfcRelConnectsElements;
   FillsVoids       : SET [0:1] OF IfcRelFillsElement;
   HasCoverings     : SET OF IfcRelCoversBldgElements;
   HasProjections   : SET OF IfcRelProjectsElement;
   HasOpenings      : SET OF IfcRelVoidsElement;
   HasPorts         : SET OF IfcRelConnectsPortToElement;
   IsConnectionRealization : SET OF IfcRelConnectsWithRealizingElements;
   ProvidesBoundaries      : SET OF IfcRelSpaceBoundary;
   ContainedInStructure    : SET [0:1] OF IfcRelContainedInSpatialStructure;
   HasControlElements      : SET [0:1] OF IfcRelFlowControlElements;
END_ENTITY;
```

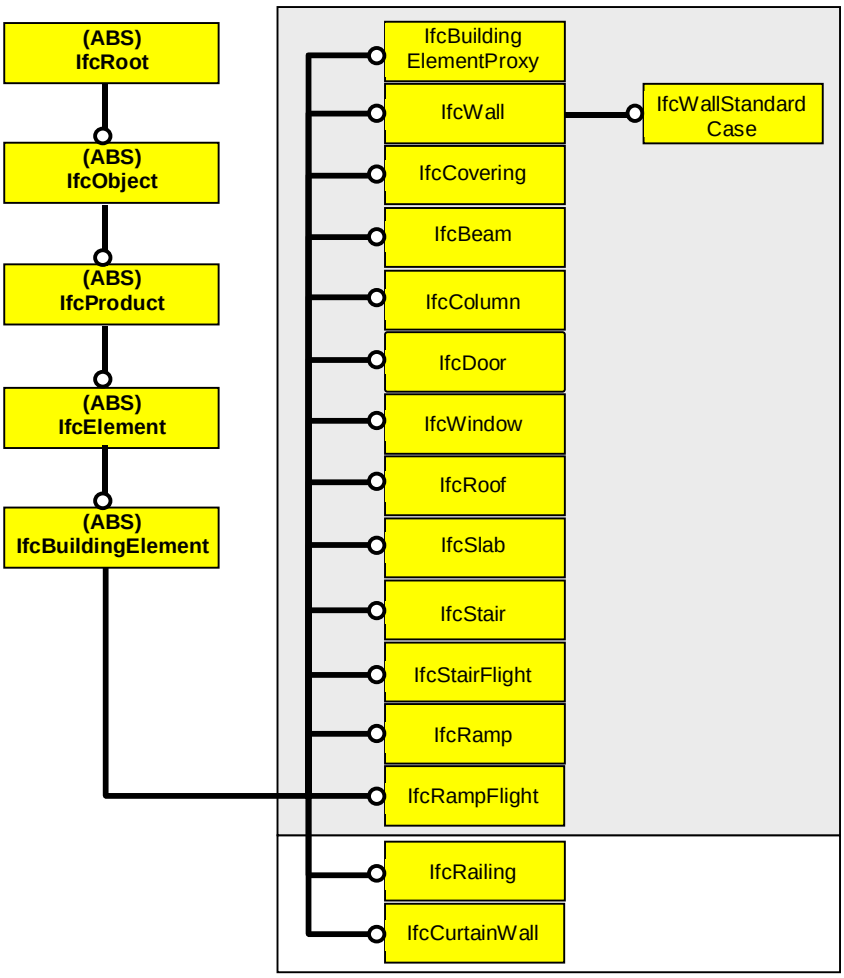


Figure 21 : Hierarchy chart of building elements

5.2 Walls

Walls are exchanged by an IFC2x file as instances of *IfcWall* or *IfcWallStandardCase*. Generally speaking, *IfcWallStandardCase* handles all cases, where a wall has a single material thickness (both in ground view and cross section) along the wall path, and *IfcWall* handles all other cases of walls.

Standard walls (supported by *IfcWallStandardCase*) include all cases for straight walls and curved walls under the following conditions:

- having a single (material) thickness along the wall path,
- having a wall path being either a straight line or a circular arc,
- may have various offsets from the wall path,
- may have a single or multiple material layers,
- may have a constant height or a varying height along the path.

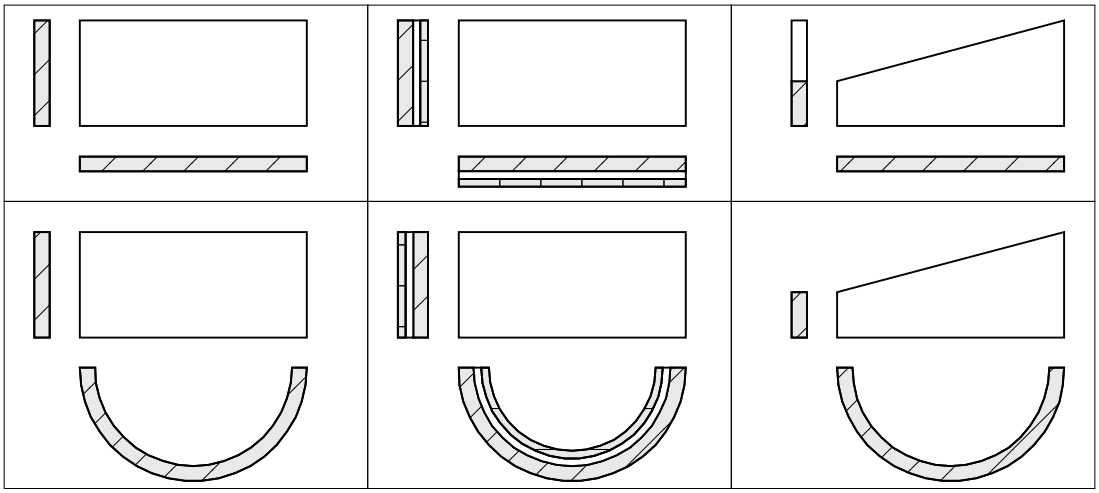


Figure 22 : Examples for standard walls (ground view, cross section and elevation)

Specific walls (supported by *IfcWall*) include all cases of walls, that are not covered by the standard walls under the following conditions:

- do not have a single (material) thickness along the wall path
  - either have a foot print with non-parallel sides, i.e. a varying material thickness along the wall path (such as a cone as foot print),
  - or have a cross section, not being rectangular, like a shear wall having an L-shape cross section
- do not have a wall path being either a straight line or a circular arc
  - either have an elliptic arc,
  - or a spline curve,
  - or any other irregular path geometry

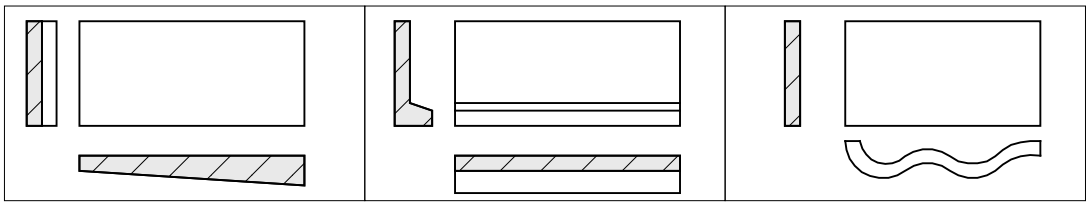


Figure 23 : Examples for specific walls (ground view, cross section and elevation)

The following units of functionality (UoF) are common for both types of wall definitions, but might be already discussed earlier, as they apply to a higher level in the hierarchy.

- placement (→ 9.1.2)
- element in space containment
- element quantities
- element connectivity
- element space boundaries
- element opening voiding

The UoF "placement" for walls requires the use of *IfcObjectPlacement*, in IFC2x views where the grid is not supported, it is further restricted to *IfcLocalPlacement*. For complete building model exchange the relative placement needs to be supported, and the spatial structure element, to which the wall is placed relatively, is normally the *IfcBuildingStorey*.

> *NOTE: If the UoF "element in space containment" is provided (and walls are required to be contained within a spatial structure element for complete building model exchange) then the IfcLocalPlacement.PlacementRelTo shall point to the same spatial structure element.*

5.2.1 Standard walls

The following units of functionality (UoF) have to be supported for standard walls, in addition to the UoF to be supported for all building elements (→ 5.1) and for all walls (→ 5.2):

- multiple shape representation (→ 9.1.3.2)
- material assignment (→ 10.2.1)

5.2.1.1 Multiple geometric representations of standard walls.

The UoF of multiple shape representation of standard walls includes (at least) two shape representations for each instance of *IfcWallStandardCase*. The first represents the wall axis, the second represents the wall body.

The wall axis is given by an instance of *IfcShapeRepresentation* with the following conventions:

- use of *IfcShapeRepresentation.RepresentationIdentifier* = "Axis"
- use of *IfcShapeRepresentation.RepresentationType* = "Curve2D"
- use of either *IfcTrimmedCurve* (with *BasisCurve* having the datatype *IfcLine*) or *IfcPolyline* for *IfcShapeRepresentation.Items[1].*

The **wall axis** also provides the reference line to relate the material layer set offset to the wall. The implicit material layer set offset line is the right edge of the first material layer (in the direction of the increasing parametric range of the geometric representation item for the wall axis).

The constraints of the *RepresentationType* "Curve2D" are discussed in 9.1.4.1.1. In particular the geometric representation item has to have the dimensionality 2. Additional rules apply due to the limitations of the standard wall axis shape:

- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the datatype of *Item[1]* shall be *IfcTrimmedCurve*, or *IfcPolyline*.

For straight walls, the *Item* is either an *IfcTrimmedCurve* with *BasisCurve* being an *IfcLine*, or an *IfcPolyline* with exactly two *Points*. For curved walls the *Item* is an *IfcTrimmedCurve* with *BasisCurve* being an *IfcCircle* only. In both cases the sense is important (for the assignment of the material layer set) and may be influenced by the *IfcTrimmedCurve.SenseAgreement* (or the sequence of *Points* for the *IfcPolyline*). In case of trimmed curves, the trimming may be given by either trimming points or by the parametric boundaries.

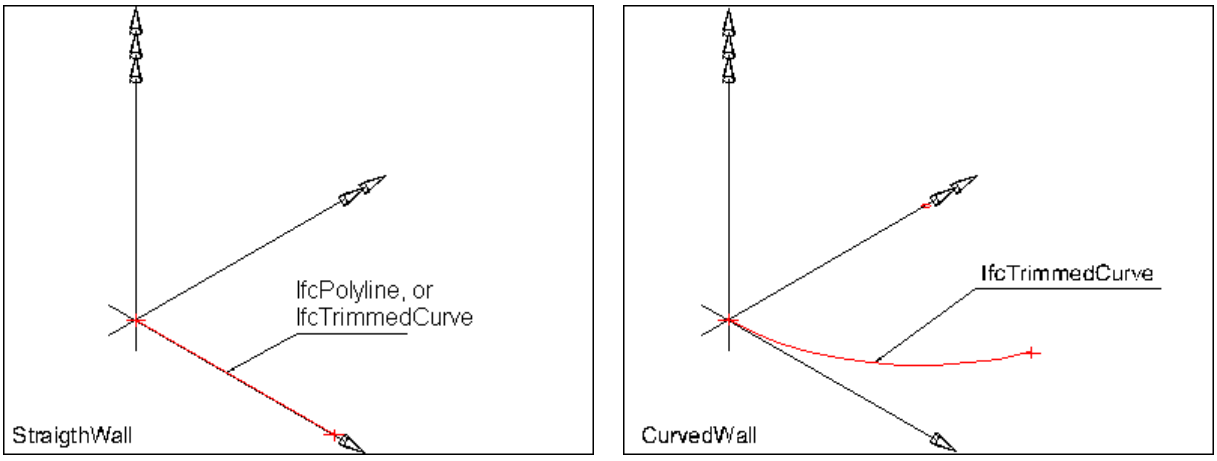


Figure 24 : Shape representation for standard wall axes

The **wall body** is given by an instance of *IfcShapeRepresentation* with the following conventions:

- use of *IfcShapeRepresentation.RepresentationIdentifier* = "Body"
- use of *IfcShapeRepresentation.RepresentationType* = "SweptSolid" or "Clipping"
- use of *IfcExtrudedAreaSolid* (with *SweptArea* of datatype *IfcArbitraryClosedProfileDef*) for *IfcShapeRepresentation.Items[1].* In case of clipped standard wall bodies, the use of *IfcBooleanClippingResult* is required (with *FirstOperand* of Type *IfcExtrudedAreaSolid*).

The wall body represents the 3D shape of the standard wall. The foot print of the wall is given as a 2D arbitrary closed curve (by *IfcArbitraryClosedProfileDef*) with the start, end and eventually existing "in between" connections already resolved.

Since the extrusion direction is vertical, there is no difference between the straight wall and the curved wall in terms of the sweep operation, both are represented by *IfcExtrudedAreaSolid* with *SweptArea* of type *IfcArbitraryClosedProfileDef* and *Depth* equal to wall height (or the maximum height for clipped wall bodies). In case non non-clipped walls having a single height, the *IfcExtrudedAreaSolid* is exchanged. In case of clipped walls (having clippings on either lower or upper boundary of the wall) the final wall body is given as a Boolean difference of the extruded solid and one or more half space solids (*IfcHalfSpaceSolid* or subtypes).

> *NOTE: It is recommended practice (although not required by the specification) to align the extrusion coordinate system (IfcSweptAreaSolid.Position) with the object coordinate system (IfcLocalPlacement.RelativePlacement) if possible.*

5.2.1.2 Standard walls with equal height

The constraints of the *RepresentationType* "SweptSolid" are discussed in 9.1.4.1.4.2. In particular the geometric representation item has to have the dimensionality 3. Additional rules apply due to the limitations of the standard wall body shape for single height, **non-clipped walls**:

- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the *IfcShapeRepresentation.RepresentationType* = "SweptSolid"
- the datatype of *Item[1]* shall be *IfcExtrudedAreaSolid*. The extrusion is vertical, i.e. perpendicular to the wall axis.

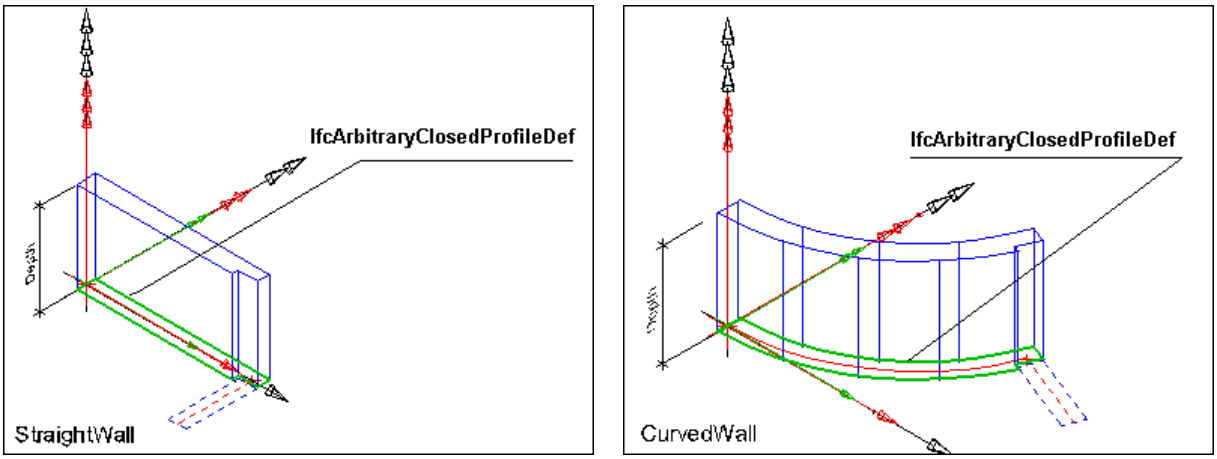


Figure 25 : Shape representation for standard, non-clipped wall bodies

In order to retrieve the wall parameter for total wall thickness and offset the material layer information have to be provided (→ 5.2.1.4). The consistency between the *IfcMaterialLayerSet.TotalThickness* and the offset between the two edges of the *IfcArbitraryClosedProfileDef.OuterCurve* has to be guaranteed by the sending application populating the IFC2x exchange model.

The wall body may be clipped on its lower and/or upper side. This e.g., provides for walls underneath sloped roofs or on top of sloped ramps. Also gable walls are generated by clipping operations. There is no restriction regarding the number of clipping surfaces, the only restriction is, that the clipping surface is a plane (and cannot be a cylindrical or any other non-planar elementary surface). The Figure 25 shows examples of standard clipped walls.

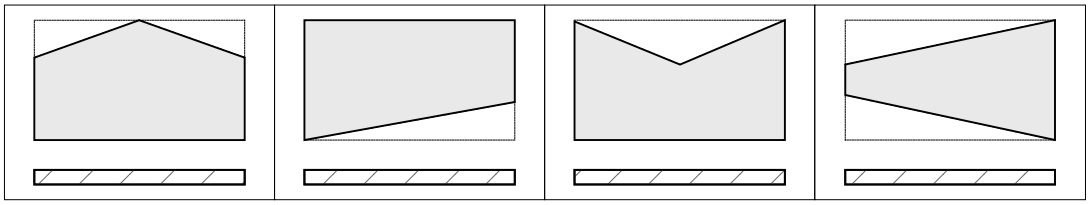


Figure 26 : Examples of standard clipped wall bodies (ground view and elevation)

5.2.1.3 Standard walls with varying height

Each clipping is represented by an *IfcBooleanClippingResult*, where the *FirstOperand* is the *IfcExtrudedAreaSolid* (of the total wall body) for the first clipping, or the *IfcBooleanClippingResult* of a previous clipping. The *SecondOperand* is an *IfcHalfSpaceSolid* (or a subtype of it). There are two different half space solids available:

- For unbounded clipping planes the *IfcHalfSpaceSolid* and the *IfcBoxedHalfSpace*
- For bounded clipping planes the *IfcPolygonalBoundedHalfSpace*[10]

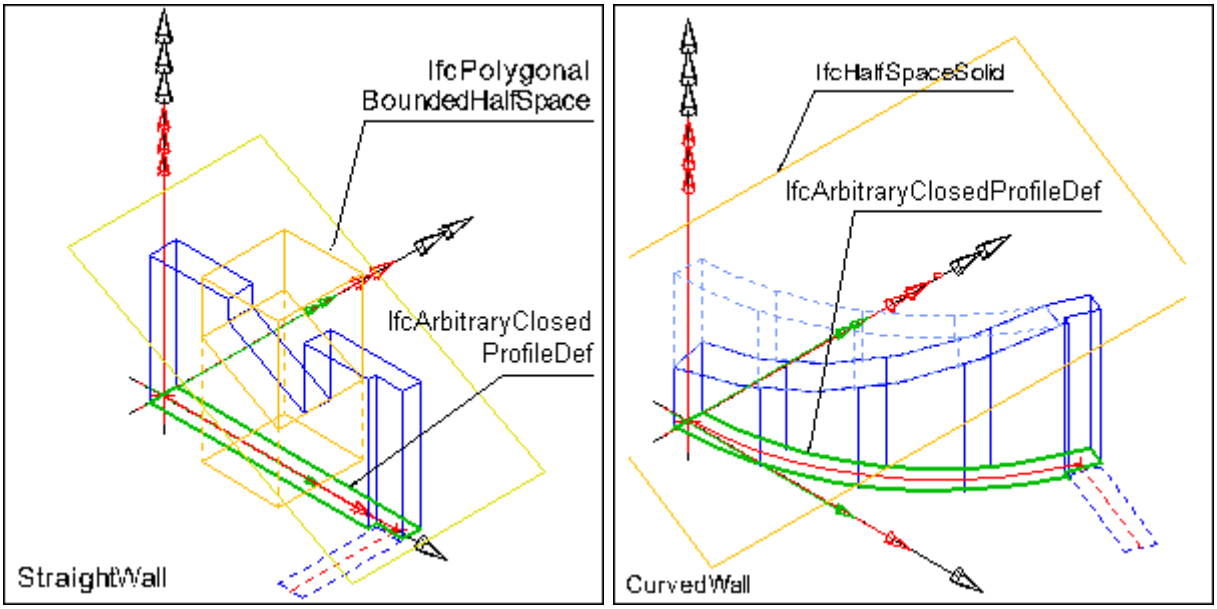


Figure 27 : Example of bounded and unbounded clipping planes

The use of *IfcBooleanClippingResult* and an example of a clipped wall, including the IFC file, is given in 9.1.4.1.4.3.1. More then one clipping can be applied to the wall body.

Additional rules apply due to the limitations of the standard wall body shape for single height, **clipped walls**:

- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the *IfcShapeRepresentation.RepresentationType* = "Clipping"
- the datatype of *Item[1]* shall be *IfcBooleanClippingResult*.

There are further restriction to the use of clipping for walls:

- only the upper and lower part of the wall shall be clipped,
- the start and end part (often referred to all start cap and end cap) shall not be clipped – these features are handled by *IfcArbitraryProfileDef* and *IfcRelConnectsPathElements*,
- the clipping shall not be used to clip parallel (or tangential) to the wall path or to create a hole into the wall body. Holes (or openings) are created by *IfcRelVoidsElement* and *IfcOpeningElement*.

5.2.1.4 Material layer set usage assignment

Each standard wall requires the provision of the material layer set definitions. This UoF is described in 10.2.1.3. The *IfcMaterialLayerSet* containing the material layer sequence and layer thickness is assigned to the *IfcWallStandardCase* by using

- *IfcMaterialLayerSetUsage* – adding the offset to the wall axis and the layering sense (allows to mirror the material layers)
- *IfcRelAssociatesMaterial* – making the link of all identical material layer set usages to instances of standard walls.

[10] Strictly speaking, this is not an half space anymore, as it does not divide the domain into exactly two parts. It is the Boolean intersection of the body of the unlimited extrusion of the polygonal bounded area and the body of the half space.

Thereby the same instance of *IfcMaterialLayerSetUsage* can be shared among several walls. As mentioned earlier, the consistency of the material layer set usage and the foot print geometry has to be guaranteed by the sending application, as this is not covered by the IFC internal EXPRESS rules.

The following examples demonstrate the use of the *IfcMaterialLayerSetUsage* for different assignments of material layer sets.

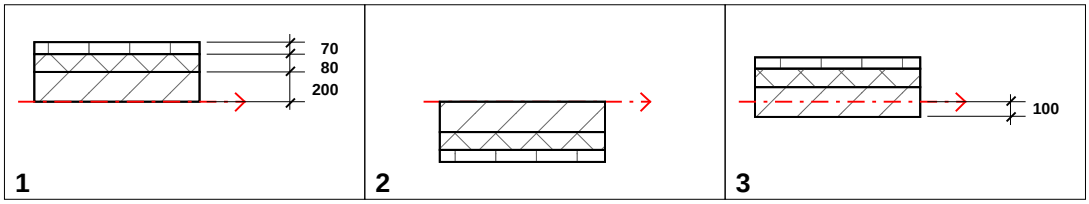


Figure 28 : Examples of material layer set usage

The IfcMaterialLayerSet is the same in all three cases, it is given as

```
#15=IFCMATERIALLAYERSET((#16,#17,#18),'Isolated outer wall type 1');
#16=IFCMATERIALLAYER(#19,200.,$);
#17=IFCMATERIALLAYER(#20,80.,$);
#18=IFCMATERIALLAYER(#21,70.,$);
#19=IFCMATERIAL('Concrete');
#20=IFCMATERIAL('Mineral wool');
#21=IFCMATERIAL('Brick');
```

However the IfcMaterialLayerSetUsage, referred to by the IfcRelAssociatesMaterial differs:

```
/* in case of wall 1 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.POSITIVE.,0.);

/* in case of wall 2 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.NEGATIVE.,0.);

/* in case of wall 3 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.POSITIVE.,-100.);
```

The value and direction of the *OffsetFromReferenceLine* is independent of the *DirectionSense* attribute of the IfcMaterialLayerSetUsage. If the *DirectionSense* is NEGATIVE, then the material layer set is mirrored at the offset line. Using the same *IfcMaterialLayerSet* as in Figure 27 the new cases look like:

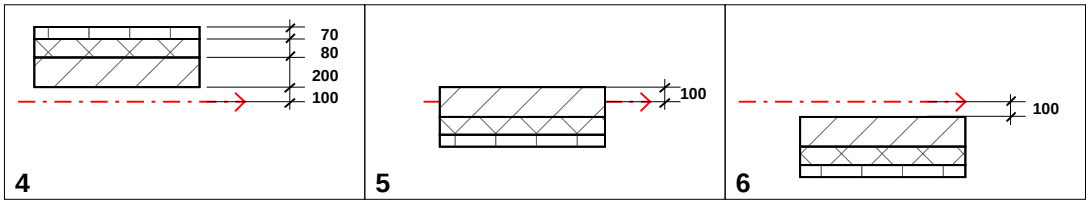


Figure 29 : Examples of direction sense agreements at material layer ser usage

Again only the IfcMaterialLayerSetUsage, referred to by the IfcRelAssociatesMaterial, differs:

```
/* in case of wall 4 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.POSITIVE.,100.);

/* in case of wall 5 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.NEGATIVE.,100.);

/* in case of wall 6 */
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#2,$,$,(#1),#14);
#14=IFCMATERIALLAYERSETUSAGE(#15,.AXIS2.,.NEGATIVE.,-100.);
```

The following additional agreements are made on the usage of the attribute *LayerSetDirection* of *IfcMaterialLayerSetUsage* for walls.

- AXIS1 means the direction of the wall axis
- AXIS2 means perpendicular to the axis in XY plane (the only one implemented in IFC2x)
  - positive at the left side, looking into the direction of the increasing parameter range of the wall axis
- AXIS3 means perpendicular to the axis in Z direction

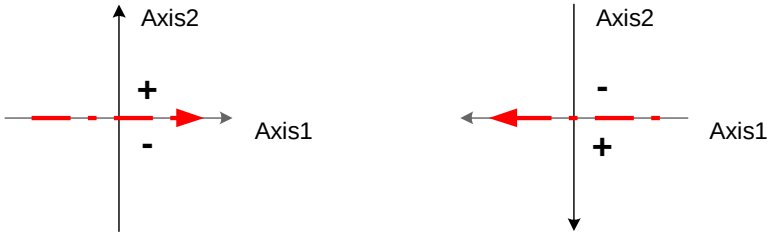


Figure 30 : Layer set directions in case of straight walls

In consequence, material layers for walls shall only be used for standard (parallel) layering, and should not be adapted for any more complex internal structures, like for elemented or stub walls.

In case of round walls the *LayerSetDirection* of *IfcMaterialLayerSetUsage* for walls has the following meaning:

- AXIS1 means the tangent to the arc, representing the wall axis, at a given point
- AXIS2 means the secant (through the center) of the arc, representing the wall axis, in XY plane (the only one implemented in IFC2x)
  - positive at the outer side of the arc, when clockwise oriented, or at the inner side of the arc, when counter-clockwise oriented
- AXIS3 means perpendicular to the axis in Z direction

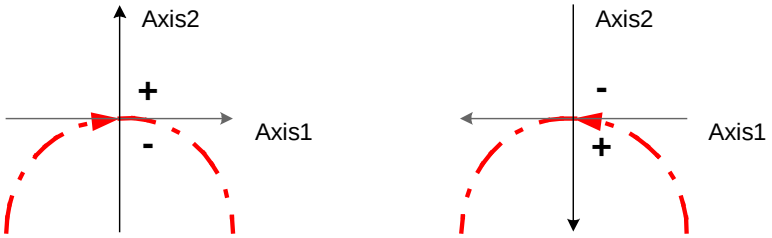


Figure 31 : Layer set directions in case of round walls

5.2.2 Specific walls

The following units of functionality (UoF) have to be supported for specific walls, in addition to the UoF to be supported for all building elements (→ 5.1) and for all walls (→ 5.2):

- multiple shape representation (→ 9.1.3.2)

A general rule applies to the shape representation of all specific walls – it shall only contain a single body (either as extrusion – *IfcSweptAreaSolid*, clipping – *IfcBooleanClippingResult*, or b-rep – *IfcFacetedBrep*). If the special wall object consists of two (geometrically apart) bodies in the creating system, it needs to be split into two instances of *IfcWall* for the export by IFC2x.

NOTE: This rule applies to all building elements that can have openings, as otherwise it would be difficult to determine to which body the opening element applies.

5.2.2.1 Multiple shape representation for specific walls (of type *IfcWall*)

The UoF of multiple shape representation of specific walls includes (at least) two shape representations for each instance of *IfcWall*. The first represents the wall axis, the second represents the wall body. The wall axis representation is described in 5.2.1.1, the following additional agreements are applicable for wall axis for specific walls:

- the provision of a wall axis is optional for walls with "Brep" shape representations of the wall body, however it should be provided, whenever it is possible and meaningful (the omission of a wall axis means that the wall geometry is so irregular, that it can only be regarded as a piece of material),
- the provision of a wall axis is mandatory for walls with "SweptSolid" shape representations of the wall body. An *IfcWall* with “SweptSolid” shape representation is used for walls with polygonal footprint, that do not have parallel sides (are a constant thickness). The extrusion direction is vertical (i.e. perpendicular to the footprint).

5.2.2.2 Specific walls with polygonal footprint

If the wall body for an *IfcWall* is given as a "SweptSolid" shape representation, the following applies. The footprint geometry is given as an *IfcArbitraryClosedProfileDef* without voids, that represents the complete footprint (including start and end caps). The *IfcExtrudedAreaSolid.Depth* represents the height of the wall. In particular:

- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the *IfcShapeRepresentation.RepresentationType* = "SweptSolid"
- the datatype of *Item[1]* shall be *IfcExtrudedAreaSolid*.
- the datatype of *IfcExtrudedAreaSolid.SweptArea* shall be *IfcArbitraryClosedProfileDef*.

The following figure shows examples for walls with polygonal footprint that should be exchanged as *IfcWall* with “SweptSolid” shape representation.

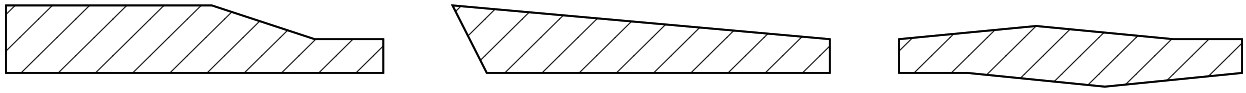


Figure 32 : Examples for polygonal walls

5.2.2.3 Specific walls with varying height

If the polygonal wall has cutting, e.g. under a roof slab, it is given as a “Clipping” shape representation, where the wall body is given by the extrusion, as explained above, and the clipping is given by *IfcHalfSpaceSolid* (or its subtypes). For the clipping part, the convention to use the Boolean operation by the *IfcBooleanClippingResult* are the same as for standard walls, see 5.2.1.3.

5.2.2.4 Specific walls with different layer heights

If the wall is defined by multiple layers, where some layers have either a different height or have different vertical offsets compared with the other layers, then this wall shall be treated as a special wall. As some systems are able to handle such wall definition parametrically, but the IFC specification for layer sets only allow for thickness to be given parametrically, such shapes have to be exported by explicit geometric shape.

An example of such a wall is given below in Figure 32.

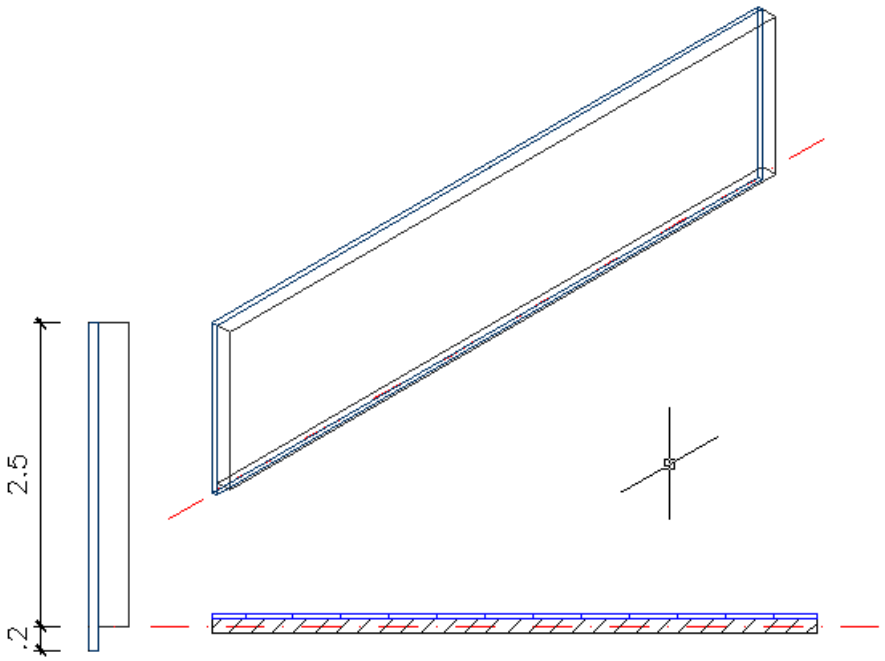


Figure 33 : Example of a wall with different layer heights

The fallback position for the exchange of such walls is to use the *RepresentationType* "Brep" for *RepresentationIdentifier* “Body” in conjunction with *RepresentationType* “Curve2D” (or “GeometricSet”) for *RepresentationIdentifier* “Axis”. Given the example in Figure 32 the exchange file should look like:

```
#77=IFCWALLSTANDARDCASE('295H6hv1z5T9dhGMC7AJX1',#16,$,$,$,#28,#37,$);
#37=IFCPRODUCTDEFINITIONSHAPE($,$,(#36,#76));
#36=IFCSHAPEREPRESENTATION(#11,'Axis','GeometricSet',(#35));
#35=IFCTRIMMEDCURVE(#32,(IFCPARAMETERVALUE(0.),#33),(IFCPARAMETERVALUE(1.),#34),.T.,.CAR
TESIAN.);
#32=IFCLINE(#29,#31);
#29=IFCCARTESIANPOINT((0.,0.));
#31=IFCVECTOR(#30,10.);
#30=IFCDIRECTION((1.,0.));
#33=IFCCARTESIANPOINT((0.,0.));
#34=IFCCARTESIANPOINT((10.,0.));
#76=IFCSHAPEREPRESENTATION(#11,'Body','Brep',(#75));
#75=IFCFACETEDBREP(#74);
#74=IFCCLOSEDSHELL((#52,#55,#58,#61,#64,#67,#70,#73));
#52=IFCFACE((#51));
#51=IFCFACEOUTERBOUND(#50,.T.);
#50=IFCPOLYLOOP((#44,#49,#47,#45));
#55=IFCFACE((#54));
#54=IFCFACEOUTERBOUND(#53,.T.);
#53=IFCPOLYLOOP((#45,#46,#43,#41,#40,#44));
#58=IFCFACE((#57));
#57=IFCFACEOUTERBOUND(#56,.T.);
#56=IFCPOLYLOOP((#48,#42,#43,#46));
#61=IFCFACE((#60));
#60=IFCFACEOUTERBOUND(#59,.T.);
#59=IFCPOLYLOOP((#47,#49,#39,#38,#42,#48));
#64=IFCFACE((#63));
#63=IFCFACEOUTERBOUND(#62,.T.);
#62=IFCPOLYLOOP((#49,#44,#40,#39));
#67=IFCFACE((#66));
#66=IFCFACEOUTERBOUND(#65,.T.);
#65=IFCPOLYLOOP((#47,#48,#46,#45));
#70=IFCFACE((#69));
#69=IFCFACEOUTERBOUND(#68,.T.);
#68=IFCPOLYLOOP((#41,#43,#42,#38));
#73=IFCFACE((#72));
#72=IFCFACEOUTERBOUND(#71,.T.);
#71=IFCPOLYLOOP((#41,#38,#39,#40));
```

```
#38=IFCCARTESIANPOINT((10.,0.2,0.));
#39=IFCCARTESIANPOINT((10.,0.12,0.));
#40=IFCCARTESIANPOINT((0.,0.12,0.));
#41=IFCCARTESIANPOINT((0.,0.2,0.));
#42=IFCCARTESIANPOINT((10.,0.2,2.7));
#43=IFCCARTESIANPOINT((0.,0.2,2.7));
#44=IFCCARTESIANPOINT((0.,0.12,0.2));
#45=IFCCARTESIANPOINT((0.,-0.12,0.2));
#46=IFCCARTESIANPOINT((0.,-0.12,2.7));
#47=IFCCARTESIANPOINT((10.,-0.12,0.2));
#48=IFCCARTESIANPOINT((10.,-0.12,2.7));
#49=IFCCARTESIANPOINT((10.,0.12,0.2));
```

Using *RepresentationType* "Clipping" for *RepresentationIdentifier* “Body” is also allowed for the exchange of such walls. In this case also the full exchange of layer set information can be consistently maintained.

5.2.2.5 Specific walls with BREP geometry

If the wall body for an *IfcWall* is given as a "Brep" shape representation, the following applies. The Brep geometry already resolves all modifications and features of the wall (such as clippings of the lower or upper side of the wall, or the start and end connections), beside the cut-out of openings. Openings are given (common to all walls) by the *IfcOpeningElement* and the *IfcRelVoidsElement*. This means, that no further Boolean operations (beside for the opening) shall be applied to the Brep wall body, or there are no clipping or other Boolean operations, as expressed by *IfcBooleanResult* or *IfcBooleanClippingResult*.

The constraints of the *RepresentationType* "Brep" are discussed in 9.1.4.1.4.1. In particular the geometric representation item has to have the dimensionality 3. Additional rules apply due to the limitations of the specific wall body shape.

- the *IfcShapeRepresentation.RepresentationType* = "Brep"
- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the datatype of *Item[1]* shall be *IfcFacetedBrep*.

5.2.2.6 Fallback for compatibility with earlier versions of IFC wall geometry

As a fallback to maintain a level of downward compatibility the wall body can also be given by a "SweptSolid" shape representation, using horizontal extrusion (along the wall axis).

> *NOTE: Horizontally extruded specific walls of type IfcWall shall only be used in conjunction with earlier versions of IFC as they do not provide the same functionality as IfcWallStandardCase (particular for start and end caps). The use of IfcWallStandardCase is strongly recommended for implementations of IFC2x.*

The constraints of the *RepresentationType* "SweptSolid" are discussed in 9.1.4.1.4.2. In particular the geometric representation item has to have the dimensionality 3. Additional rules apply due to the limitations of the specific wall body shape for single height, non-clipped walls, with horizontal extrusions along the wall axis.

- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the *IfcShapeRepresentation.RepresentationType* = "SweptSolid"
- the datatype of *Item[1]* shall be *IfcExtrudedAreaSolid*.
- the datatype of *IfcExtrudedAreaSolid.SweptArea* shall be *IfcRectangleProfileDef*.

5.2.3 Special cases of wall configurations

The following shows special cases for walls and guidelines on how to solve them. Some CAD systems keep a wall as a single entity, even if it is totally split into two parts, e.g. by placing a column in the middle of it, or by X-type wall connections.

> *NOTE: In the IFC model, a wall has to have a single body, it is not allowed (according to the current view definitions) to export a wall having two totally separated wall bodies (either as swept solids, or as BREP). Therefore the sending system has to split the wall into two walls before exporting it to IFC.*

The following figures show some examples, where a wall is split into two parts:

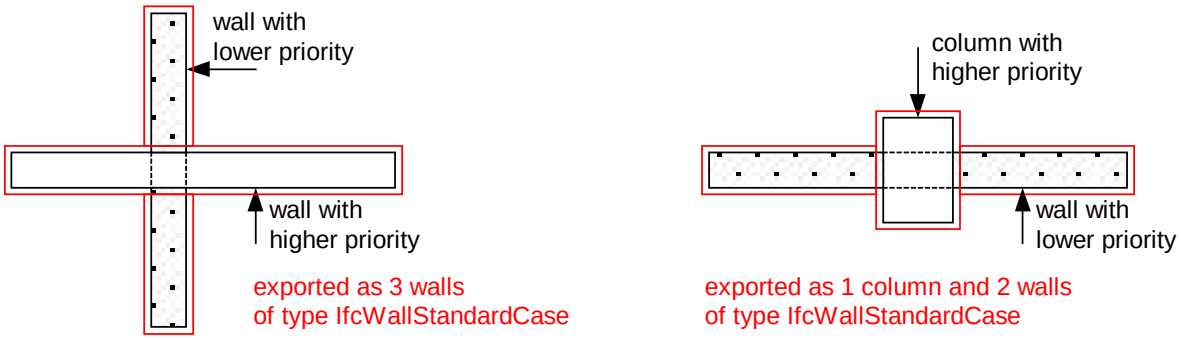


Figure 34 : Special cases of walls split into two parts

In these cases the single original wall is exported as two walls. In addition the connection information should be exported, in case of wall to wall connection by *IfcRelConnectsPathElements*, in case of wall to column connections as *IfcRelConnectsElements*.

Other cases include:

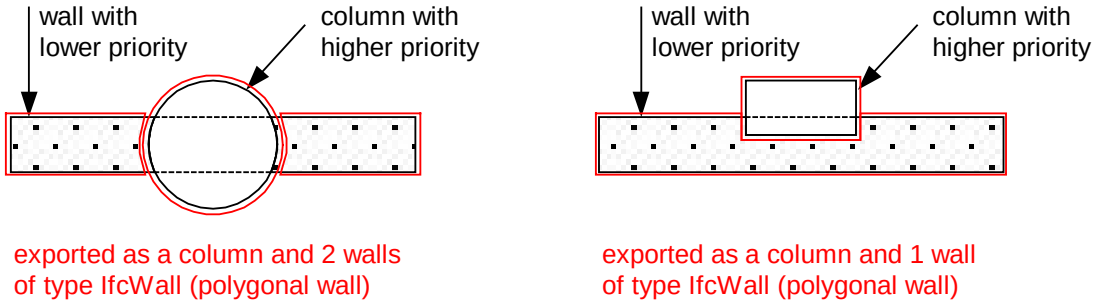


Figure 35 : Special cases of walls split or cut out by columns

The polygonal walls are exported as *IfcWall*, following the conventions as explained in 5.2.2.2. In all cases there is the fall back position to export the wall body as a BREP shape representation as explained in 5.2.2.4.

5.3 Openings

Openings are exchanged in IFC2x files as instances of *IfcOpeningElement*. An *IfcOpeningElement* forms a void, which is created within another element. This is primarily independent from the kind of element, which is voided (like a wall opening, or a slab opening, or an opening in a roof) and also from the fact, whether the void goes through the whole thickness of the element (a normal opening) are whether it is only partially voided (like a recess, chase or trench).

In addition to the UoF provided for all elements (see 5), each opening has three additional UoF:

- Element Opening Voiding (see this section),
- Opening Element Filling (see section 5.3.3),
- Opening Representation (see this section).

The relationship to the element which is voided by the opening is provided by *IfcRelVoidsElement*, the provision of this relationship is mandatory for each instance of *IfcOpeningElement*. An opening may have a filling (a window or a door) inserted, in this case it is referenced by *IfcRelFillsElement*. The Figure 35 shows the relations between elements and openings:

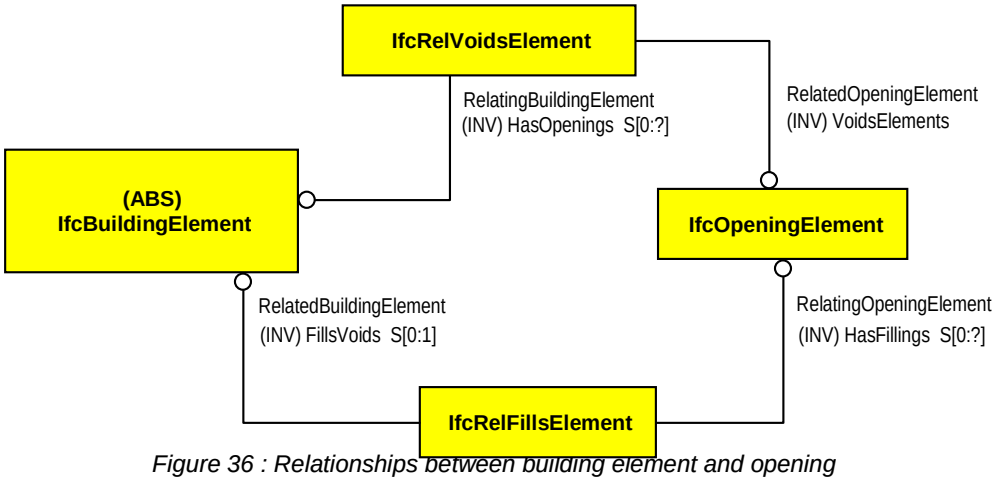


Figure 36 : Relationships between building element and opening

Within the view definitions and implementation guidelines restrictions are defined about the subtypes of *IfcBuildingElement* which can play the role of being voided or being a filling. Normally only *IfcWindow* and *IfcDoor* (as subtypes of *IfcBuildingElement*) are used as fillings. *IfcWall, IfcSlab*, but also *IfcBeam, IfcColumn, IfcStairFlight, IfcRampFlight, IfcCovering* and *IfcBuildingElementProxy* should be normally supported as elements that can be voided by openings.

The opening local placement shall be given relative to the local placement of the building element, which is voided[11]. Therefore the *IfcLocalPlacement.PlacementRelTo* shall point to the local placement of the voided building element.

The body of the opening is placed relative to the local placement of the opening. The standard geometric representation of the body is given by:

- use of *IfcShapeRepresentation.RepresentationIdentifier* = "Body"
- use of *IfcShapeRepresentation.RepresentationType* = "SweptSolid"
- use of *IfcExtrudedAreaSolid* (with *SweptArea* of datatype *IfcProfileDef* and its subtypes) for *IfcShapeRepresentation.Items[1].*

The *IfcExtrudedAreaSolid.Position* needs to encounter for the default extrusion direction, being along the local z-Axis of this placement of the extruded solid. If a parameterized subtype of *IfcProfileDef* is used to describe the opening profile, the 2D placement of the profile is given relative to the XY plane of the *IfcExtrudedAreaSolid.Position*.

The conventions about the orientations of the local placement and extruded area solid position depend on the kind of opening. However a general convention applies to all:

- The 2D profile placement – the Position attribute of all parameterized subtypes of *IfcProfileDef* shall not be rotated, i.e. the *IfcAxisPlacement2D.RefDirection* is always [1.,0.] (or omitted, then it defaults to [1.,0.]).

The following example shows a wall opening, where the local placement of the opening is calculated as intermediate coordinate system between the IfcExtrudedAreaSolid.Position and the local placement of the wall. In this example the IfcExtrudedAreaSolid.Position is placed centric (for the position of the rectangular opening profile at IfcAxisPlacement2D.location = [0.,0.], however it is also permissible to place the IfcExtrudedAreaSolid.Position at a corner and move the profile through its 2D placement.

[11] Only within view definitions, that generally foresee global placements for all elements, this requirement should be waived.

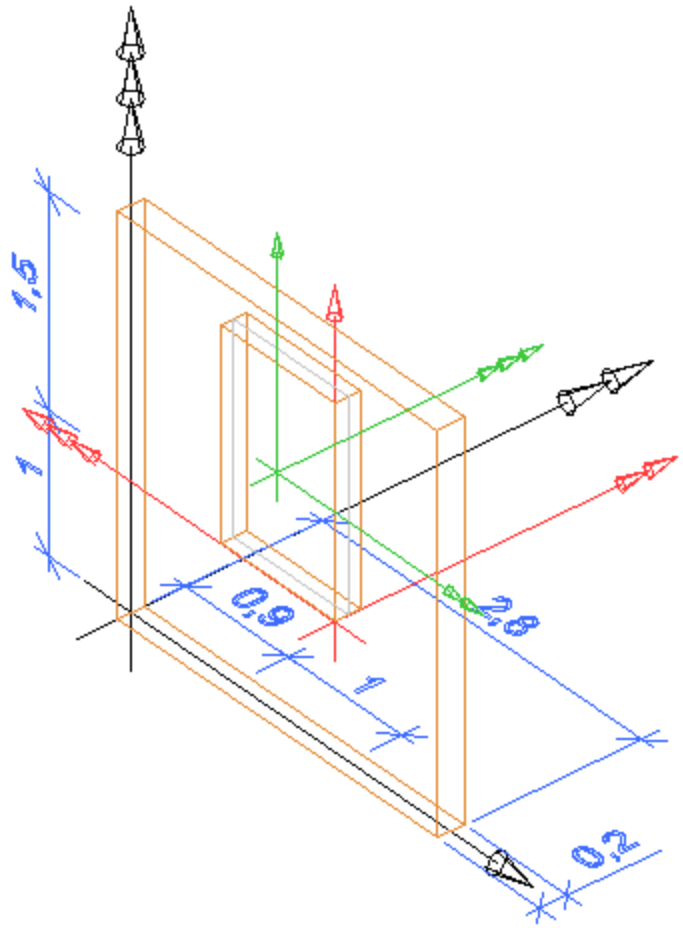


Figure 37 : Example of an opening in a wall

```
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.,1.,0.));
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11,#13,#211));

/* opening with relationship, local placement (red in figure) rel. to the wall
#81=IFCOPENINGELEMENT('2DVz9Ik7nDE8UYM00UgOTP',#2,$,'',$,#80,#75,$);
#82=IFCRELVOIDSELEMENT('0CFN5X3K519evNPGT30ZIp',#2,$,$,#1,#81);
#80=IFCLOCALPLACEMENT(#3,#79);
#76=IFCCARTESIANPOINT((1.9,-0.1,1.));
#77=IFCDIRECTION((-1.,0.,0.));
#78=IFCDIRECTION((0.,0.,1.));
#79=IFCAXIS2PLACEMENT3D(#76,#77,#78);
#75=IFCPRODUCTDEFINITIONSHAPE($,$,(#74));

/* opening geometry with extruded area solid placement (green in figure)
#74=IFCSHAPEREPRESENTATION(#111,'Body','SweptSolid',(#72));
#72=IFCEXTRUDEDAREASOLID(#65,#70,#71,0.2);
#65=IFCRECTANGLEPROFILEDEF(.AREA.,$,#64,1.5,1.);
#64=IFCAXIS2PLACEMENT2D(#62,#63);
#62=IFCCARTESIANPOINT((0.,0.));
#63=IFCDIRECTION((1.,0.));
#70=IFCAXIS2PLACEMENT3D(#67,#68,#69);
#67=IFCCARTESIANPOINT((0.75,0.,0.5));
#68=IFCDIRECTION((0.,1.,0.));
#69=IFCDIRECTION((1.,0.,0.));
#71=IFCDIRECTION((0.,0.,1.));
```

The advanced geometric representation of the opening body is given by the B-rep representation:

- the *IfcShapeRepresentation.RepresentationType* = "Brep"
- only a single *Item* shall exists within the SET of *IfcShapeRepresentation.Items*
- the datatype of *Item[1]* shall be *IfcFacetedBrep*.

Since B-rep representations can not be translated back into the parametric description of profile based opening geometry and should therefore only be used for rare exception cases.

5.3.1 Openings in walls

Openings in walls are inserted by placing the *IfcOpeningElement* into the local coordinate system of the *IfcWall*. There are no additional conventions on placing the local coordinate system, although it is preferable to use a recognizable style, e.g. having the opening z-axis pointing into the same direction as the wall z-axis.

The *IfcOpeningElement* can be used to create openings and niches within a wall. The attribute *IfcOpeningElement.ObjectType* shall be used to differentiate the usage, for openings:

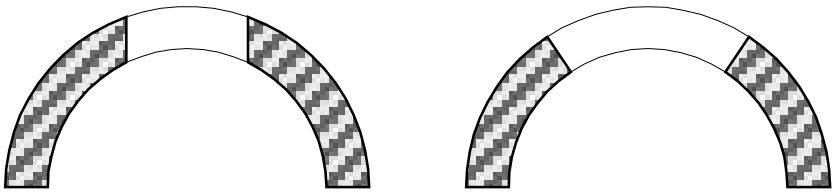
- *ObjectType* = 'Opening' (or omitted) indicates an extrusion perpendicular to the wall face. In this case the extrusion depth (*IfcExtrudedAreaSolid.Depth*) can be assumed unlimited – i.e. it may safely be increased to create a true opening.
- *ObjectType* = 'OpeningAdvanced' indicates an extrusion parallel to the wall face (in the direction of the wall extrusion). In this case the extrusion depth (*IfcExtrudedAreaSolid.Depth*) is significant.

In contrary the placement of the opening body, the *IfcExtrudedAreaSolid.Position*, is required to follow a particular pattern. The *IfcExtrudedAreaSolid.Position* shall:

openings with parallel reveals openings with shaped reveals

Figure 38 : different ways to create openings in straight walls

- for straight walls (see Figure 37):
  - if the opening reveals are parallel to each other – using *ObjectType* = 'Opening'
    - the x-axis shall point into the direction of the height of the opening,
    - the y-axis shall point into the direction of the width of the opening
    - the z-axis shall be the extrusion direction (perpendicular to the wall face)
  - if the opening reveals are not parallel to each other (e.g. having a shape like a partial setback) – using *ObjectType* = 'OpeningAdvanced' – note this should only be used, if the shaped reveals are restricted to the left and right side, not to the above and below. A different mechanism to exchange this more complicated geometry of openings is discussed in 5.3.3.
    - the x-axis shall point into the direction of the length of the opening,
    - the y-axis shall point into the direction of the width of the opening
    - the z-axis shall be the extrusion direction (parallel to wall face – identical to wall extrusion)
  -



openings with parallel reveals openings with radial reveals

Figure 39 : different ways to create openings in round walls

- for curved walls (see Figure 40):
  - if the opening reveals are parallel to each other – using *ObjectType* = 'Opening'
    - the x-axis shall point into the direction of the height of the opening,
    - the y-axis shall point into the direction of the width of the opening

  - the z-axis shall be the extrusion direction (perpendicular to the wall face)
- if the opening reveals are radial (perpendicular to the tangents at the edges) using *ObjectType* = 'OpeningAdvanced'
  - the XY plan shall be coplanar with the wall base (of the wall profile be extruded),
  - the z-axis shall point into the direction of the height of the opening (parallel to wall face – identical to wall extrusion)

The Position of the parameterized profile, e.g. the *IfcRectangleProfileDef.Position* should not provide a rotation, but may translate the profile within the XY plane of the extrusion body coordinate system.

5.3.1.1 Openings in straight standard walls

The following examples show how the local placements and the opening geometry is defined for a rectangular opening within a straight wall. It should be noted, that there is more than one possibility to define the transformations. Strict requirements are only given to the orientations of the extrusion body position (see above) and that the 2D placement of the profile shall not be rotated. The local placement of the opening could have any orientation (see also Figure 36).

The following conventions are used in the figures: red is the local placement of the opening, green is the position of the extruded body, and blue is the position of the profile.

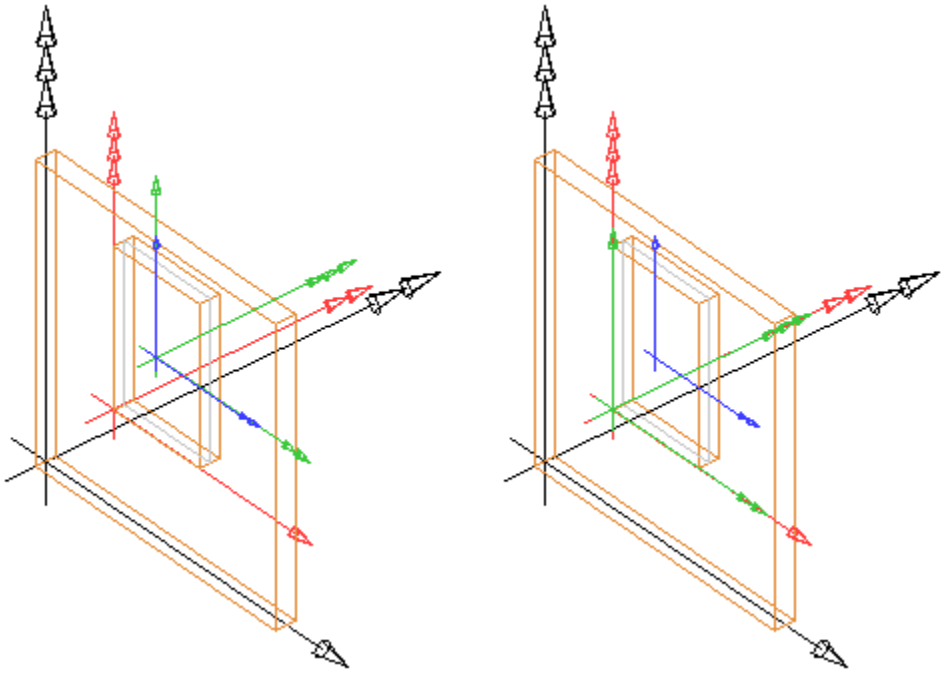


Figure 40 : Examples of an opening in a wall

Taken the previous example, the following cutouts from the *.ifc files would reflect the exchange, both (and any other local placement of the opening) are equally valid:

```
/* opening with relationship and local placement rel. to the wall */
#81=IFCOPENINGELEMENT('2DVz9Ik7nDE8UYM00UgOTP',#2,$,'',$,#80,#75,$);
#82=IFCRELVOIDSELEMENT('0CFN5X3K519evNPGT30ZIp',#2,$,$,#1,#81);
#80=IFCLOCALPLACEMENT(#3,#79);
#76=IFCCARTESIANPOINT((0.9,-0.1,1.));
#77=IFCDIRECTION((0.,0.,1.));
#78=IFCDIRECTION((1.,0.,0.));
#79=IFCAXIS2PLACEMENT3D(#76,#77,#78);
#75=IFCPRODUCTDEFINITIONSHAPE($,$,(#74));
```

```
/* opening geometry with extruded area solid placement */
/* for the left example in Figure 65 */
#74=IFCSHAPEREPRESENTATION(#111,'Body','SweptSolid',(#72));
#72=IFCEXTRUDEDAREASOLID(#65,#70,#71,0.2);
#65=IFCRECTANGLEPROFILEDEF(.AREA.,$,#64,1.5,1.);
#64=IFCAXIS2PLACEMENT2D(#62,#63);
#62=IFCCARTESIANPOINT((0.,0.));
#63=IFCDIRECTION((1.,0.));
#70=IFCAXIS2PLACEMENT3D(#67,#68,#69);
#67=IFCCARTESIANPOINT((0.5,0.,0.75));
#68=IFCDIRECTION((0.,1.,0.));
#69=IFCDIRECTION((0.,0.,1.));
#71=IFCDIRECTION((0.,0.,1.));

/* for the right example Figure 39 */
#74=IFCSHAPEREPRESENTATION(#111,'Body','SweptSolid',(#72));
#72=IFCEXTRUDEDAREASOLID(#65,#70,#71,0.2);
#65=IFCRECTANGLEPROFILEDEF(.AREA.,$,#64,1.5,1.);
#64=IFCAXIS2PLACEMENT2D(#62,#63);
#62=IFCCARTESIANPOINT((0.75,0.5));
#63=IFCDIRECTION((1.,0.));
#70=IFCAXIS2PLACEMENT3D(#67,#68,#69);
#67=IFCCARTESIANPOINT((0.,0.,0.));
#68=IFCDIRECTION((0.,1.,0.));
#69=IFCDIRECTION((1.,0.,0.));
#71=IFCDIRECTION((0.,0.,1.));
```

5.3.1.2 Openings in round walls

The standard definition of openings in round walls is given by the same definition of the *IfcExtrudedAreaSolid*, as for straight walls. The local placement of the opening is given by providing one axis being parallel to the tangent of the wall face at the center of the opening (or as a ray through the corners of the opening at the same side and height.

The position of the *IfcExtrudedAreaSolid* is given taking the above mentioned requirements for the x/y/z directions into account – the extrusion will then create parallel reveals on each opposite sides of the opening.

The following example shows an opening in a round wall, where the local placement of the opening is calculated as an intermediate coordinate system between the IfcExtrudedAreaSolid.Position and the local placement of the wall. In this example the IfcExtrudedAreaSolid.Position is placed centric (for the position of the rectangular opening profile at IfcAxisPlacement2D.location = [0.,0.]), however it is also permissible to place the IfcExtrudedAreaSolid.Position at a corner and move the profile through its 2D placement. The IfcOpeningElement.ObjectPlacement is rotated by the tangent of the wall at the middle of the opening.

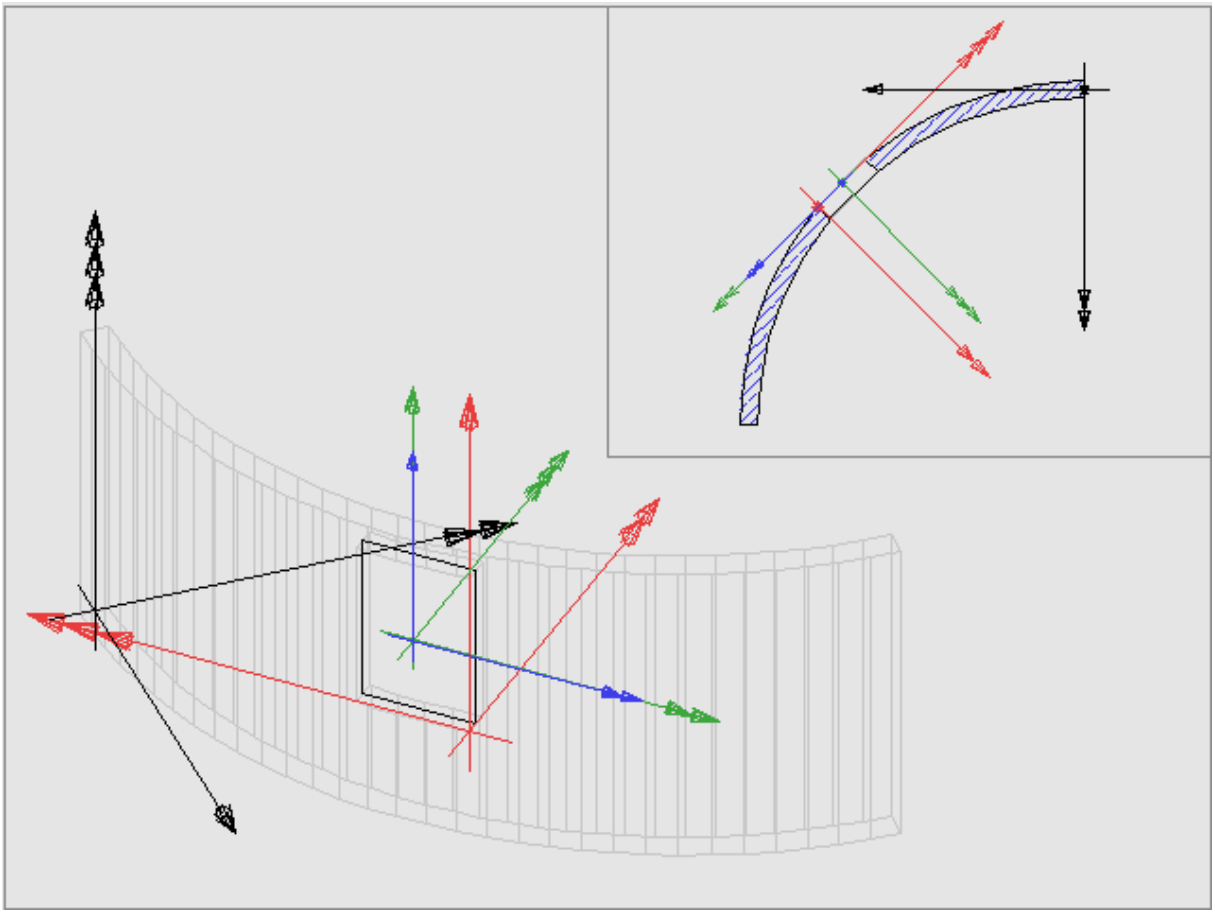


Figure 41 : Opening inserted into a round wall

It should be noted again, that there are several possible variations of the local placement for the opening and the position of the extrusion body. The following part of the .ifc file shows the exchange of openings in round walls:

```
/* definition of the wall and local placement */
#83=IFCWALLSTANDARDCASE('3fFqW2FMDFBwdjJLgDdZvV',#16,$,'',$,#31,#40,$);
#31=IFCLOCALPLACEMENT($,#30);
#30=IFCAXIS2PLACEMENT3D(#27,#28,#29);
#29=IFCDIRECTION((-1.,0.,0.));
#28=IFCDIRECTION((0.,0.,1.));
#27=IFCCARTESIANPOINT((7.,9.,0.));

/* definition of the opening, relation to the wall and local placement */
#108=IFCOPENINGELEMENT('3pqcEFeSnAyffagvfFhnEo',#16,$,'',$,#107,#102,$);
#109=IFCRELVOIDSELEMENT('1XARQjIFH3_O8ENp3HGBc8',#16,$,$,#83,#108);
#107=IFCLOCALPLACEMENT(#31,#106);
#106=IFCAXIS2PLACEMENT3D(#103,#104,#105);
#103=IFCCARTESIANPOINT((4.048141905757483,1.658964875429065,1.));
#104=IFCDIRECTION((-0.7071067811865475,-0.7071067811865476,0.));
#105=IFCDIRECTION((0.,0.,1.));

/* definition of the shape of the opening */
#102=IFCPRODUCTDEFINITIONSHAPE($,$,(#101));
#101=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#99));
#99=IFCEXTRUDEDAREASOLID(#92,#97,#98,0.5);
#92=IFCRECTANGLEPROFILEDEF(.AREA.,$,#91,1.5,1.);
#91=IFCAXIS2PLACEMENT2D(#89,#90);
#89=IFCCARTESIANPOINT((0.,0.));
#90=IFCDIRECTION((1.,0.));
#97=IFCAXIS2PLACEMENT3D(#94,#95,#96);
#94=IFCCARTESIANPOINT((0.75,0.,0.5));
#95=IFCDIRECTION((0.,1.,0.));
#96=IFCDIRECTION((1.,0.,0.));
#98=IFCDIRECTION((0.,0.,1.));
```

Similarly to the case for openings in straight walls, also for openings in round walls there are different possibilities for the coordinate systems. Below is one additional example:

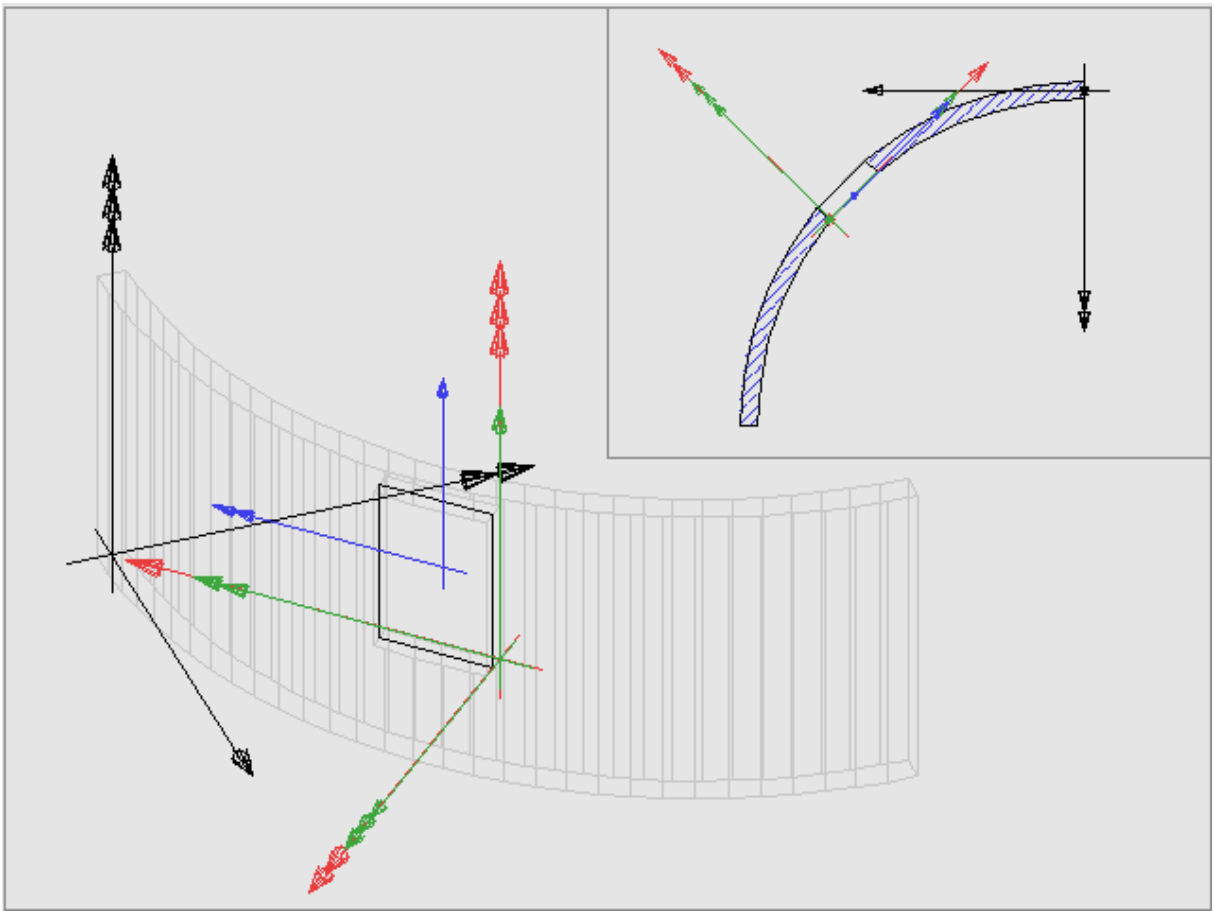


Figure 42 : Opening inserted into a round wall with different local placement

The following extract of the .ifc file shows the different settings of the placement and position attributes. It is based on the same example as the previous file, however the local placement is different and the offset for the centric insertion of the opening profile is created by offsetting the 2D profile placement.

```
/* definition of the opening, relation to the wall and local placement */
#108=IFCOPENINGELEMENT('3pqcEFeSnAyffagvfFhnEo',#16,$,'',$,#107,#102,$);
#109=IFCRELVOIDSELEMENT('1XARQjIFH3_O8ENp3HGBc8',#16,$,$,#83,#108);
#107=IFCLOCALPLACEMENT(#31,#106);
#106=IFCAXIS2PLACEMENT3D(#103,#104,#105);
#103=IFCCARTESIANPOINT((3.78216679,1.92493999,1.));
#104=IFCDIRECTION((0.,0.,1.));
#105=IFCDIRECTION((-0.7071067811865475,-0.7071067811865476,0.));

/* definition of the shape of the opening */
#102=IFCPRODUCTDEFINITIONSHAPE($,$,(#101));
#101=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#99));
#99=IFCEXTRUDEDAREASOLID(#92,#97,#98,0.5);
#92=IFCRECTANGLEPROFILEDEF(.AREA.,$,#91,1.5,1.);
#91=IFCAXIS2PLACEMENT2D(#89,#90);
#89=IFCCARTESIANPOINT((0.75,0.5));
#90=IFCDIRECTION((1.,0.));
#97=IFCAXIS2PLACEMENT3D(#94,#95,#96);
#94=IFCCARTESIANPOINT((0.,0.,0.));
#95=IFCDIRECTION((0.,1.,0.));
#96=IFCDIRECTION((0.,0.,1.));
#98=IFCDIRECTION((0.,0.,1.));
```

The definition of the opening geometry by an extrusion perpendicular to the wall face creates an opening with the reveals being parallel to each other. This is only one example for openings in round walls, a different example is inserting the openings with reveals along the radii.

Those openings where the reveals shall be radial (see Figure 38) need to be exchanged as openings with *ObjectType* = 'OpeningAdvanced', and:

- extrusion profile of type *IfcArbitraryClosedProfileDef* being positioned coplanar to the wall base and extruded along the wall extrusion direction.

It should be noted that only rectangular openings with radial reveals can be exchanged this way, any other opening shape with radial reveals need to be exchanged using B-rep geometry.

5.3.2 Niches and recesses

Niches and recesses (or chase, trenches, etc.) are described within IFC2x as *IfcOpeningElement*, where (for standard Swept Solid geometric representation) the extrusion depth, the *IfcExtrudedAreaSolid.Depth* determines the thickness of the recess (in other cases of B-rep geometric representation, the exact body of the opening geometry). In order to determine the difference to a "normal" opening, all instances of an *IfcOpeningElement*, representing a niche or recess, should be clearly labeled by the *IfcOpeningElement.ObjectType* = "Recess".

```
#108=IFCOPENINGELEMENT('3pqcEFeSnAyffagvfFhnEo',#16,$,$,'Recess',#107,#102,$);
```

The geometric representation of niches and recesses follows the representation of openings, the convention of applying the local coordinate system and the position of the extrusion body as well. The position relative to the wall thickness and the direction of the extrusion is essential to determine the correct side where the niche or recess appears.

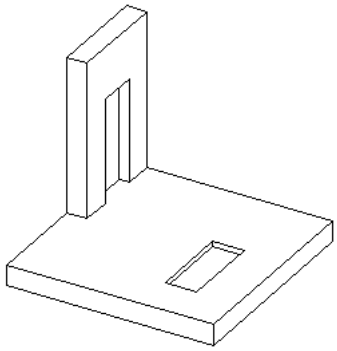


Figure 43 : Recess and niche in wall and slab

The following example, which is based on Figure 36 shows the definition of a niche within a straight wall. The IfcOpeningElement is labeled as 'Recess' and positioned according to a calculated local placement. The extrusion body has an extrusion of 0.1m, which is the thickness of the niche. The position of the extrusion body is centric to the rectangular profile, and the profile placement is [0.,0.]. Again, different positions (as given for the openings in Figure 39) are possible.

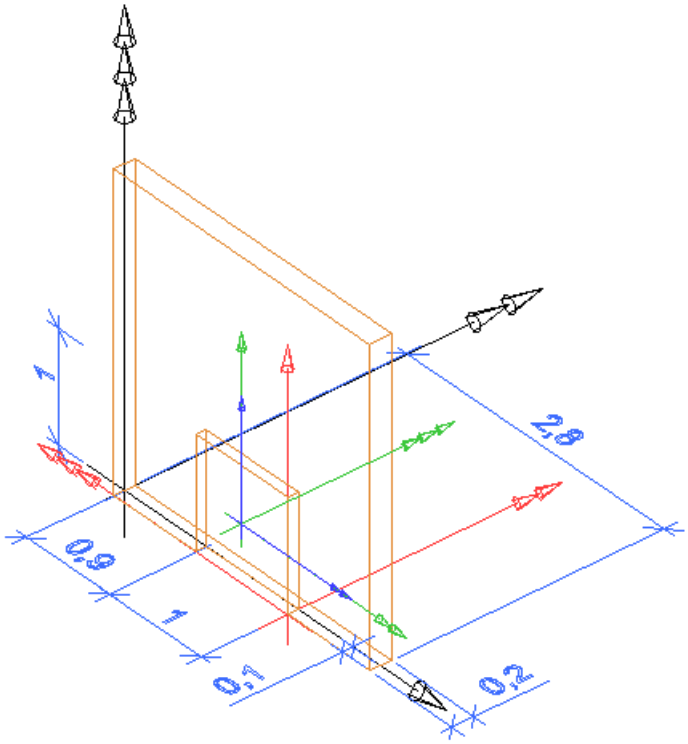


Figure 44 : Example of a niche in a straight wall

```
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.,1.,0.));
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11,#13,#211));

/* opening with relationship, local placement (red in figure) rel. to the wall
#81=IFCOPENINGELEMENT('2DVz9Ik7nDE8UYM00UgOTP',#2,$,$,'Recess',#80,#75,$);
#82=IFCRELVOIDSELEMENT('0CFN5X3K519evNPGT30ZIp',#2,$,$,#1,#81);
#80=IFCLOCALPLACEMENT(#3,#79);
#76=IFCCARTESIANPOINT((1.9,-0.1,0.));
#77=IFCDIRECTION((-1.,0.,0.));
#78=IFCDIRECTION((0.,0.,1.));
#79=IFCAXIS2PLACEMENT3D(#76,#77,#78);
#75=IFCPRODUCTDEFINITIONSHAPE($,$,(#74));

/* opening geometry with extruded area solid placement (green in figure)
#74=IFCSHAPEREPRESENTATION(#111,'Body','SweptSolid',(#72));
#72=IFCEXTRUDEDAREASOLID(#65,#70,#71,0.1);
#65=IFCRECTANGLEPROFILEDEF(.AREA.,$,#64,1.,1.);
#64=IFCAXIS2PLACEMENT2D(#62,#63);
#62=IFCCARTESIANPOINT((0.,0.));
#63=IFCDIRECTION((1.,0.));
#70=IFCAXIS2PLACEMENT3D(#67,#68,#69);
#67=IFCCARTESIANPOINT((0.5,0.,0.5));
#68=IFCDIRECTION((0.,1.,0.));
#69=IFCDIRECTION((1.,0.,0.));
#71=IFCDIRECTION((0.,0.,1.));
```

5.3.3 Openings and recesses combined

Some openings have a more complicated shape, either by using shaped reveals, or by using two separate shapes for the front and back part of the opening. These openings should be exchanged as:

- a combination of opening and recess
- an opening with BREP shape representation

Examples of this type of openings include:

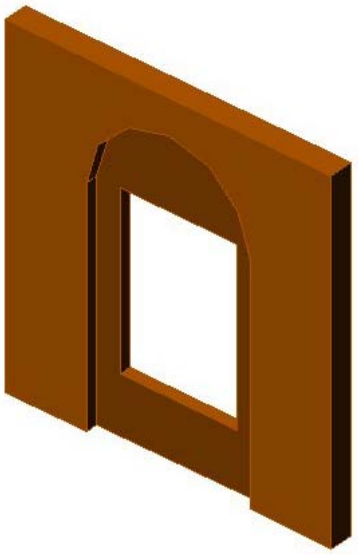


Figure 45 : Example for special opening types

Such openings shall be exported as one *IfcOpening* with the *ObjectType* “Opening” (the smaller part, creating the full opening), and one *IfcOpening* with the *ObjectType* “Recess” (the bigger, additional part).

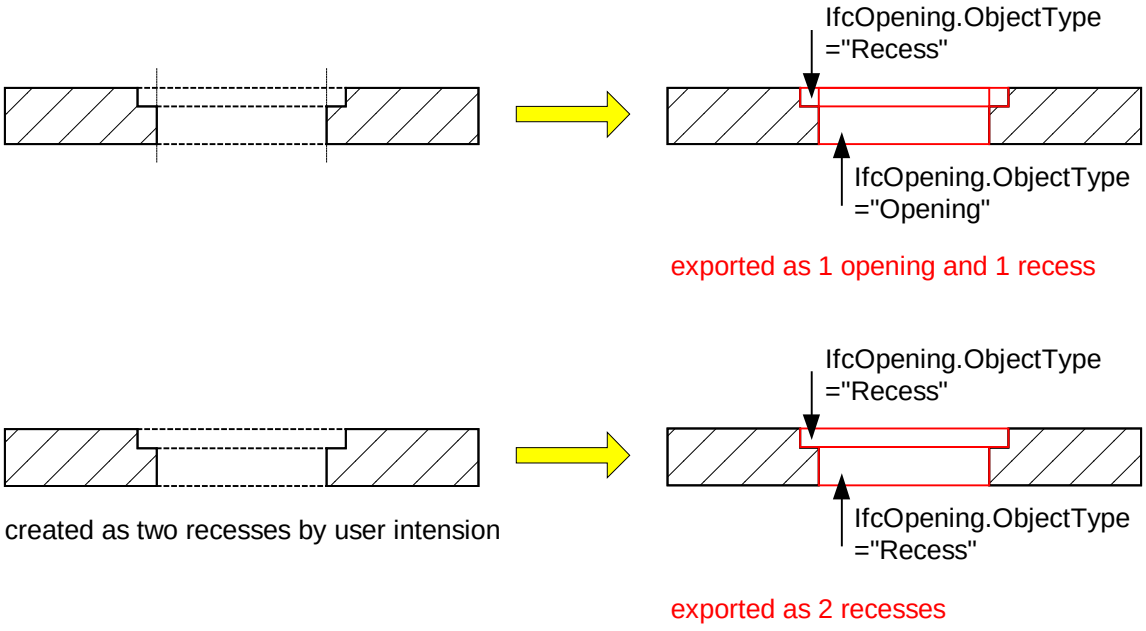


Figure 46 : Exchange of special opening types

The default exchange is 1 opening and 1 recess. However if the user has created the special opening in the original system by two recesses (or subtraction bodies) by its own intention, the exchange of two recesses is permitted. Both, the opening and the recess, has the “SweptSolid” shape representation, extruded perpendicular to the wall face.

More complex opening types should be exchanged as openings with BREP shape representations. This is the e.g. the case, if the reveals are oblique.

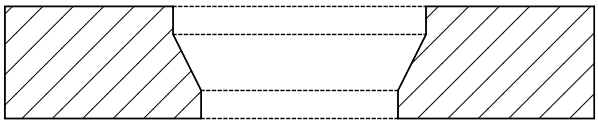


Figure 47 : Example of a special opening type to be exchanged as BREP

Note: If an opening is exchanged with BREP shape representation, or as two recesses, and it has a door or a window as a filling, the window or door geometry has to be exchanged with BREP shape representation as well.

5.3.4 Opening in slabs

Openings in slabs are inserted by placing the *IfcOpeningElement* into the local coordinate system of the *IfcSlab*. There are no additional conventions on placing the local coordinate system, although it is preferable to use a recognizable style, e.g. having the opening z-axis pointing into the same direction as the slab z-axis.

The *IfcOpeningElement* can be used to create openings and niches within a slab. The attribute *IfcOpeningElement.ObjectType* shall be used to differentiate the usage, for openings:

- *ObjectType* = 'Opening' (or omitted) indicates an extrusion perpendicular to the slab footprint (i.e. using the same extrusion direction as the swept solid of the slab body).
- *ObjectType* = 'OpeningAdvanced' indicates an extrusion non-perpendicular to the slab footprint (i.e. using a different extrusion direction as the swept solid of the slab body).
- *ObjectType* = 'Recess' indicates an extrusion perpendicular to the slab footprint (i.e. using the same extrusion direction as the swept solid of the slab body). The depth of the extrusion is less than the thickness of the slab, thereby creating a recess rather then an opening.

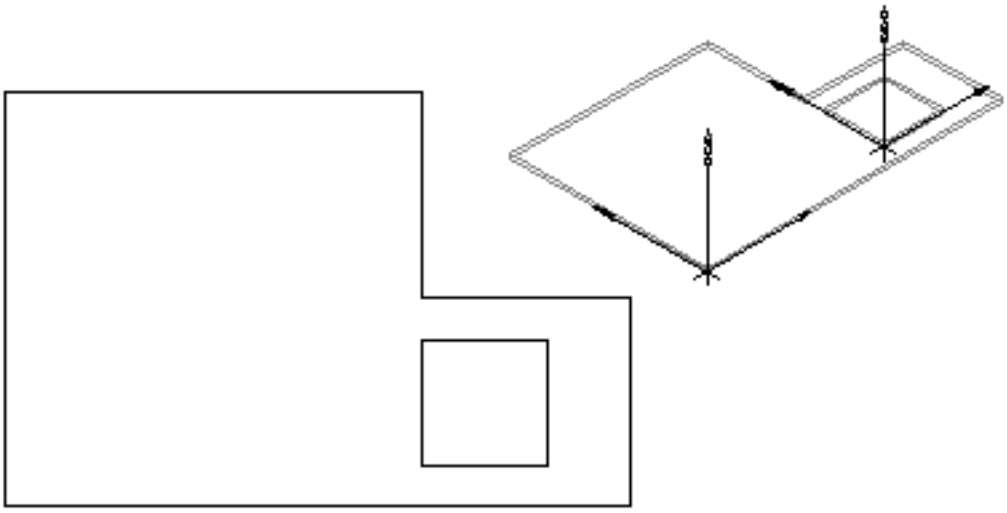


Figure 48 : Example of slab opening

Example:

The following example shows the definition of an opening within a slab. Here the rectangular opening is exported as IfcArbitraryCloseProfileDef with an OuterCurve of type IfcPolyline, another possibility would be IfcRectangleProfileDef. The ObjectType attribute at the entity IfcOpeningElement is omitted, therefore it defaults to 'Opening'.

```
#70=IFCRELVOIDSELEMENT('3Zn4jNSWfEZgsTKxbdGa8R',#16,$,$,#47,#69);
 /* definition of the slab */
 #47=IFCSLAB('0UjlDeHfTA5BfnW65$ZKfb',#16,$,'',$,#28,#46,$,.FLOOR.);
  #28=IFCLOCALPLACEMENT(#22,#27);
   /* local placement data of building storey skipped */
   #27=IFCAXIS2PLACEMENT3D(#24,#25,#26);
    #24=IFCCARTESIANPOINT((5.,5.,0.));
    #25=IFCDIRECTION((0.,0.,1.));
    #26=IFCDIRECTION((1.,0.,0.));
  #46=IFCPRODUCTDEFINITIONSHAPE($,$,(#45));
   #45=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#43));
    #43=IFCEXTRUDEDAREASOLID(#37,#41,#42,0.2);
     #37=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#36);
      #36=IFCPOLYLINE((#29,#30,#31,#32,#33,#34,#35));
       #29=IFCCARTESIANPOINT((0.,0.));
       #30=IFCCARTESIANPOINT((15.,0.));
       #31=IFCCARTESIANPOINT((15.,5.));
       #32=IFCCARTESIANPOINT((10.,5.));
```

```
      #33=IFCCARTESIANPOINT((10.,10.));
      #34=IFCCARTESIANPOINT((0.,10.));
      #35=IFCCARTESIANPOINT((0.,0.));
    #41=IFCAXIS2PLACEMENT3D(#38,#39,#40);
     #38=IFCCARTESIANPOINT((0.,0.,0.));
     #39=IFCDIRECTION((0.,0.,1.));
     #40=IFCDIRECTION((1.,0.,0.));
    #42=IFCDIRECTION((0.,0.,1.));
/* definition of the slab opening */
#69=IFCOPENINGELEMENT('1JXjB0oPbDTe7dzWWyuO6x',#16,$,$,$,#52,#68,$);
 #52=IFCLOCALPLACEMENT(#27,#51);
  /* placement relative to the slab placement */
  #51=IFCAXIS2PLACEMENT3D(#48,#49,#50);
   #48=IFCCARTESIANPOINT((10.,1.,0.));
   #49=IFCDIRECTION((0.,0.,1.));
   #50=IFCDIRECTION((1.,0.,0.));
 #68=IFCPRODUCTDEFINITIONSHAPE($,$,(#67));
  #67=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#65));
   #65=IFCEXTRUDEDAREASOLID(#59,#63,#64,0.2);
    #59=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#58);
     #58=IFCPOLYLINE((#53,#54,#55,#56,#57));
      #53=IFCCARTESIANPOINT((0.,3.));
      #54=IFCCARTESIANPOINT((0.,0.));
      #55=IFCCARTESIANPOINT((3.,0.));
      #56=IFCCARTESIANPOINT((3.,3.));
      #57=IFCCARTESIANPOINT((0.,3.));
    #63=IFCAXIS2PLACEMENT3D(#60,#61,#62);
     #60=IFCCARTESIANPOINT((0.,0.,0.));
     #61=IFCDIRECTION((0.,0.,1.));
     #62=IFCDIRECTION((1.,0.,0.));
    #64=IFCDIRECTION((0.,0.,1.));
```

5.4 Fillings (doors and windows)

Fillings (as the common classification of doors and windows) are exchanged in IFC2x files as instances of *IfcDoor* and *IfcWindow*. Each filling can be placed within an opening – actually each door and window within a wall requires the existence of an opening, in which it is placed.

Doors and windows are usually manufactured elements, which are inserted as occurrences of its type. This concept is used to define doors and windows within IFC2x. Each occurrence of a window or door is exchanged as an instance of *IfcDoor* or *IfcWindow*. The type information is provided by *IfcDoorStyle* or *IfcWindowStyle*, which aggregates the properties of doors and windows, and (if given) the 3D (B-rep) geometry of that door or window style.

All doors and windows, being of the same style, share the same instance of *IfcDoorStyle* or *IfcWindowStyle*. Figure 48 shows all relevant IFC2x entities needed to define an instance of door or window.

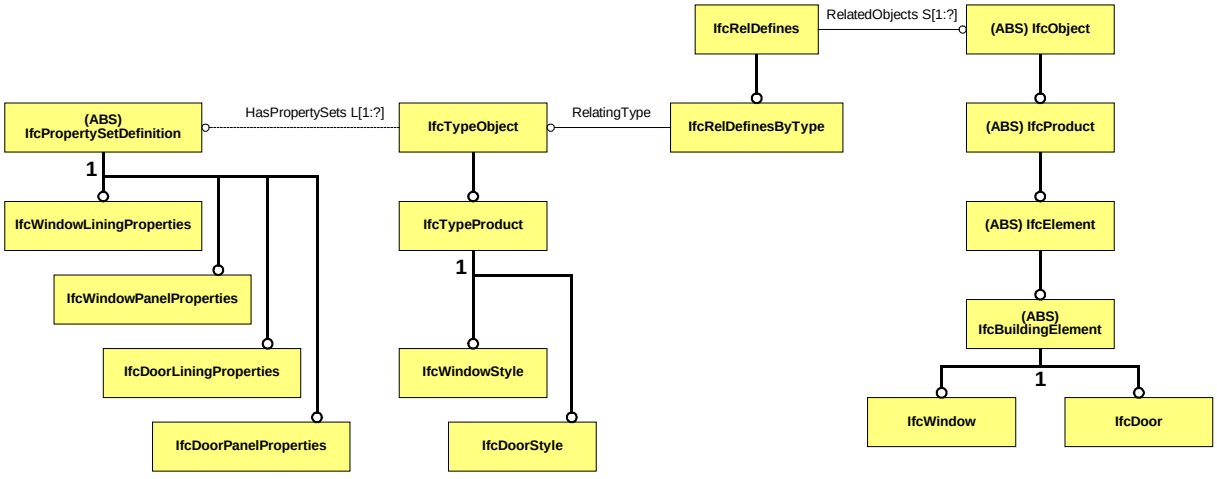


Figure 49 : Definitions for IfcDoor and IfcWindow

In addition to the UoF provided for all elements (see section 5), each filling has two additional UoF:

- Opening Element Filling (see further within this section),
- Opening Representation (see further within this section).

5.4.1 Doors

Doors are defined for the IFC2x data sharing as individual instances of *IfcDoor* on the occurrence side and shared instances of *IfcDoorStyle*, including one instance of *IfcDoorLiningProperties* and at least one instance of *IfcDoorPanelProperties* at the type side. The following is defined:

by *IfcDoor*

- The position of the door relative to the opening and wall (particularly the offset position to the wall edge),
- The opening direction of the door (swing in, swing out)
- The overall width and height of the door
- The individual geometric (B-rep) representation (if given), as a mapped item (transformation)

by *IfcDoorStyle*

- The name and description of the style
- The operation type of the door (this includes also the distinction of left vs. right opening swing doors, as these are two different types from a manufacturing point of view)
- The construction type (for information purposes only)
- and two indications of the use of the (potentially) attached shared geometric (B-rep) representation (whether it overrides eventually given additional parameters and whether the attached geometry is sizable)

by *IfcDoorLiningProperties*

- (as minimum) depth and thickness
- (additional) threshold, chasing and transom parameter, if applicable to the door style

by *IfcDoorPanelProperties*

- (as minimum) depth and width of the panel
- operation type and position of the panel (the latter is important, if a multi panel door is exchanged, in this case two or three (depending on the number of panels) instances of *IfcDoorPanelProperties* are attached to the instance of *IfcDoorStyle*.

Note: The use of IfcDoorStyle is mandatory for each occurrence of IfcDoor. Therefore each instance of IfcDoor has to have a link IfcDoor ← IsDefinedBy – IfcRelDefinesByType – RelatingType → IfcDoorStyle. The provision of door lining and panel properties, i.e. IfcDoorLiningProperties and IfcDoorPanelProperties is optional (if not otherwise requested by a view definition), if a given property is not given in the IFC data, it can be replaced by a default value at the receiving system.

There are strict conventions on the use of the local placement for *IfcDoor*.

Location:

- the local origin of the door placement shall be the insertion point of the door lining (all length parameter of the lining and panel are positive and shall be applied along the positive side of the y-axis.

Axis direction:

- the x-axis shall point into the direction of the panel (when closed), this is parallel to the wall path or the tangent for curved walls in the middle of the opening,
- the y-axis shall point into the direction of the door opening
- the z-axis shall point vertical

The axis direction and the operation type determines the four possibilities of the operation of (single) swing doors.

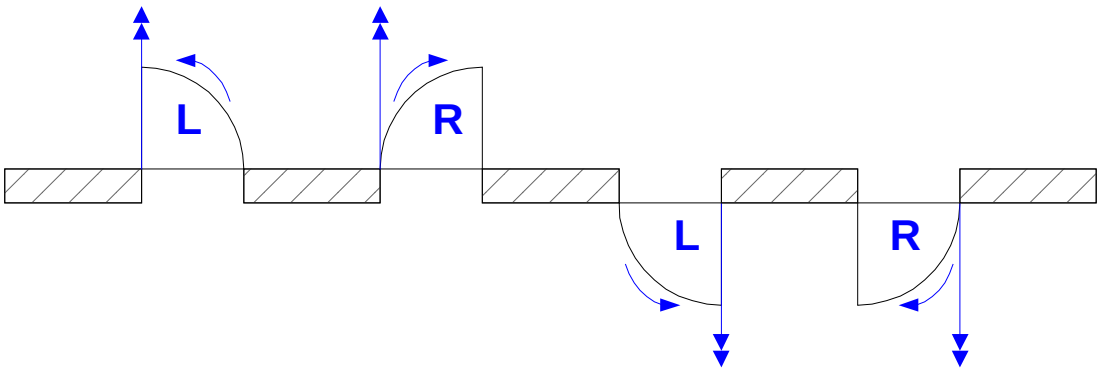


Figure 50 : Door opening and hinge directions

The following example shows a standard single swing door, which is opening to the left. The lining depth is 0.12m (within a 0.2m thick wall) and it is positioned centric within the wall. Therefore the local placement is moved by 0.16 {(0.2-0.12)/2+0.12}. The panel thickness is 0.05. The overall height is 2m and the width is 1m.

The door swings along the positive y axis of the local placement of the door. It opens to the left (seen in the direction of the swing opening) – this is determined by the OperationStyle SINGLE_SWING_LEFT. The door style name is 'Standard'.

Notation: black = local placement of wall, grey = local placement of opening, green = local placement of door.

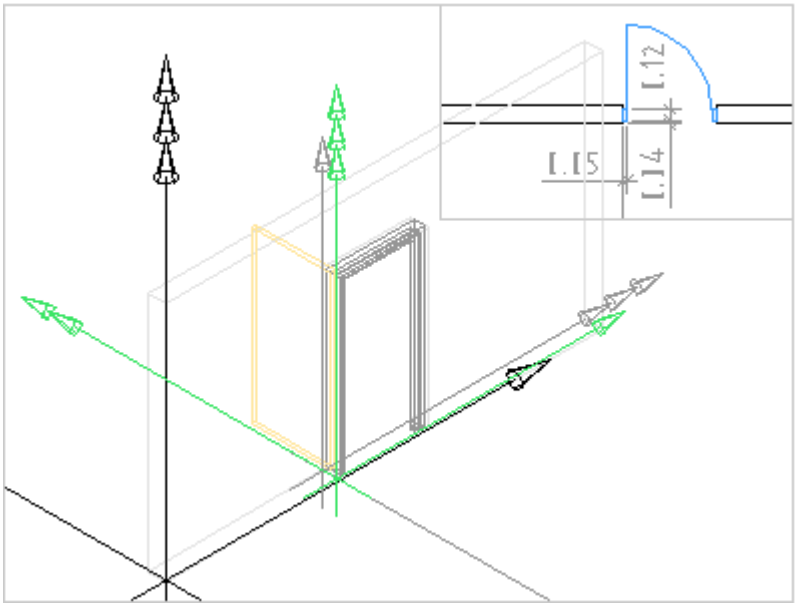


Figure 51 : A standard single swing door

```
/* wall definition */
#56=IFCWALLSTANDARDCASE('3PncBLdwj24hl30CxJLWqb',#16,$,'',$,#31,#40,$);
#31=IFCLOCALPLACEMENT($,#30);
#30=IFCAXIS2PLACEMENT3D(#27,#28,#29);
#27=IFCCARTESIANPOINT((0.,0.,0.));
#28=IFCDIRECTION((0.,0.,1.));
#29=IFCDIRECTION((1.,0.,0.));

/* opening definition */
#81=IFCOPENINGELEMENT('1kGh5XRdX08vrpvxcoOz00',#16,$,'',$,#80,#75,$);
#80=IFCLOCALPLACEMENT(#31,#79);
#79=IFCAXIS2PLACEMENT3D(#76,#77,#78);
#76=IFCCARTESIANPOINT((2.,0.2,0.));
#77=IFCDIRECTION((1.,0.,0.));
#78=IFCDIRECTION((0.,0.,1.));
#82=IFCRELVOIDSELEMENT('0fIqzgCAbAswweLFwz9uOV',#16,$,$,#56,#81);
```

```
/* door definition */
#91=IFCDOOR('3cCAHy3JbB3eDyXfruy7HK',#16,$,'',$,#90,$,$,2.,1.);
#90=IFCLOCALPLACEMENT(#80,#89);
#89=IFCAXIS2PLACEMENT3D(#86,#87,#88);
#86=IFCCARTESIANPOINT((0.,0.16,0.));
#87=IFCDIRECTION((1.,0.,0.));
#88=IFCDIRECTION((0.,0.,1.));
#92=IFCRELFILLSELEMENT('19RQHSNCP0shWm8ynf19RK',#16,$,$,#81,#91);

/* door type definitions */
#93=IFCRELDEFINESBYTYPE('3w12FoWQH2X9TTevy830PA',#16,$,$,(#91),#85);
#85=IFCDOORSTYLE('0Cq4HglY1DeAYBj0RB2XmO',#16,'Standard',$,'',(#83,#84),$,'',.SINGLE_SWI
NG_LEFT.,.NOTDEFINED.,.T.,.F.);
#83=IFCDOORLININGPROPERTIES('3efP6cLufD_8xUjPp5u36X',#16,$,$,0.12,0.05,$,$,$,$,$,$,$,$
);
#84=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenxUhv',#16,$,$,0.05,.SWINGING.,1.,.LEFT.,$)
;
```

The following should describe the three other principle positions and opening directions of the single swing door. The example about shows an outward swinging left hinged door, the other three possibilities are outward swinging right hinged door, inward swinging left hinged door, and inward swinging right hinged door.

The flipping between a left hinged and a right hinged door is done by changing the assigned *IfcDoorStyle.OperationType*, from SINGLE_SWING_LEFT to SINGLE_SWING_RIGTH[12]. The flipping between the outward and inward swinging direction is done by rotating the local placement of the door.

The following three examples of a single swing door are all based on the base example as in Figure 50, only the swing and hinge direction changes. The parts of the .ifc files highlight the changed parameters.

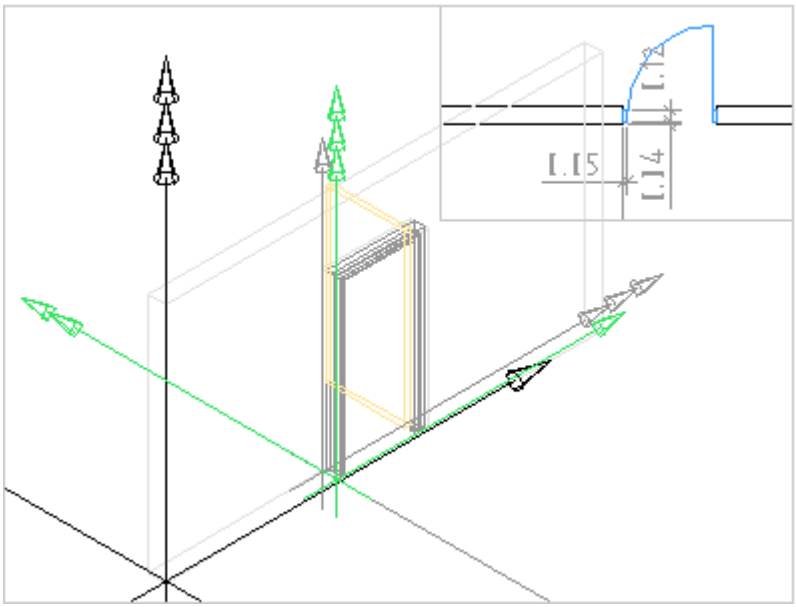


Figure 52 : single swing door (outward, right side hinge)

```
/* door definition */
#91=IFCDOOR('3cCAHy3JbB3eDyXfruy7HK',#16,$,'',$,#90,$,$,2.,1.);
#90=IFCLOCALPLACEMENT(#80,#89);
#89=IFCAXIS2PLACEMENT3D(#86,#87,#88);
#86=IFCCARTESIANPOINT((0.,0.16,0.));
#87=IFCDIRECTION((1.,0.,0.));
#88=IFCDIRECTION((0.,0.,1.));
#92=IFCRELFILLSELEMENT('19RQHSNCP0shWm8ynf19RK',#16,$,$,#81,#91);

/* door type definitions */
#93=IFCRELDEFINESBYTYPE('3w12FoWQH2X9TTevy830PA',#16,$,$,(#91),#85);
#85=IFCDOORSTYLE('0Cq4HglY1DeAYBj0RB2XmO',#16,'Standard',$,'',(#83,#84),$,'',
```

[12] This solution was taken, since strictly speaking a left and a right hinged door are two separate styles, since manufactured differently.

```
    .SINGLE_SWING_RIGHT.,.NOTDEFINED.,.T.,.F.);
#83=IFCDOORLININGPROPERTIES('3efP6cLufD_8xUjPp5u36X',#16,$,$,0.12,0.05,$,$,$,$,$,$,$,$,$
);
#84=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenxUhv',#16,$,$,0.05,.SWINGING.,1.,.LEFT.,$)
;
```

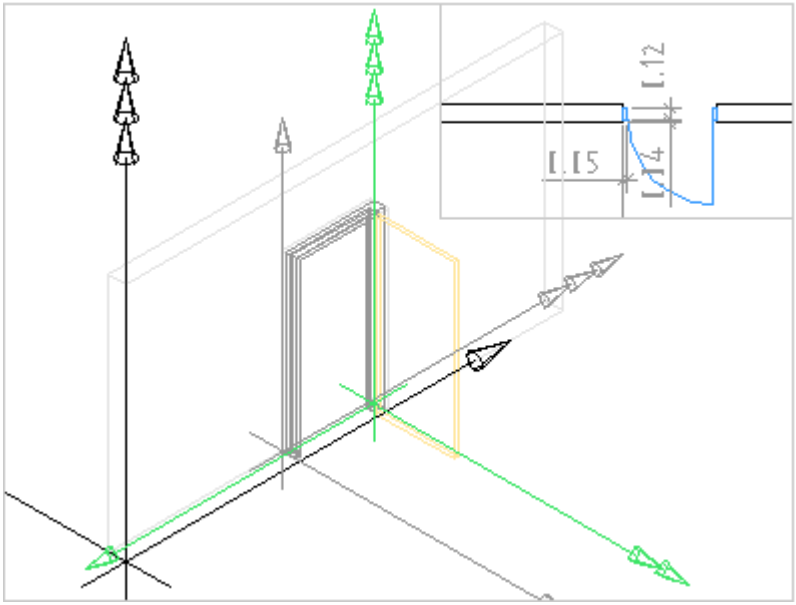


Figure 53 : single swing door (inward, left side hinge)

```
/* door definition */
#91=IFCDOOR('3cCAHy3JbB3eDyXfruy7HK',#16,$,'',$,#90,$,$,2.,1.);
#90=IFCLOCALPLACEMENT(#80,#89);
#89=IFCAXIS2PLACEMENT3D(#86,#87,#88);
#86=IFCCARTESIANPOINT((0.,0.04,1.));
#87=IFCDIRECTION((1.,0.,0.));
#88=IFCDIRECTION((0.,0.,-1.));
#92=IFCRELFILLSELEMENT('19RQHSNCP0shWm8ynf19RK',#16,$,$,#81,#91);

/* door type definitions */
#93=IFCRELDEFINESBYTYPE('3w12FoWQH2X9TTevy830PA',#16,$,$,(#91),#85);
#85=IFCDOORSTYLE('0Cq4HglY1DeAYBj0RB2XmO',#16,'Standard',$,'',(#83,#84),$,'',.SINGLE_SWI
NG_LEFT.,.NOTDEFINED.,.T.,.F.);
#83=IFCDOORLININGPROPERTIES('3efP6cLufD_8xUjPp5u36X',#16,$,$,0.12,0.05,$,$,$,$,$,$,$,$,$
);
#84=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenxUhv',#16,$,$,0.05,.SWINGING.,1.,.LEFT.,$)
;
```

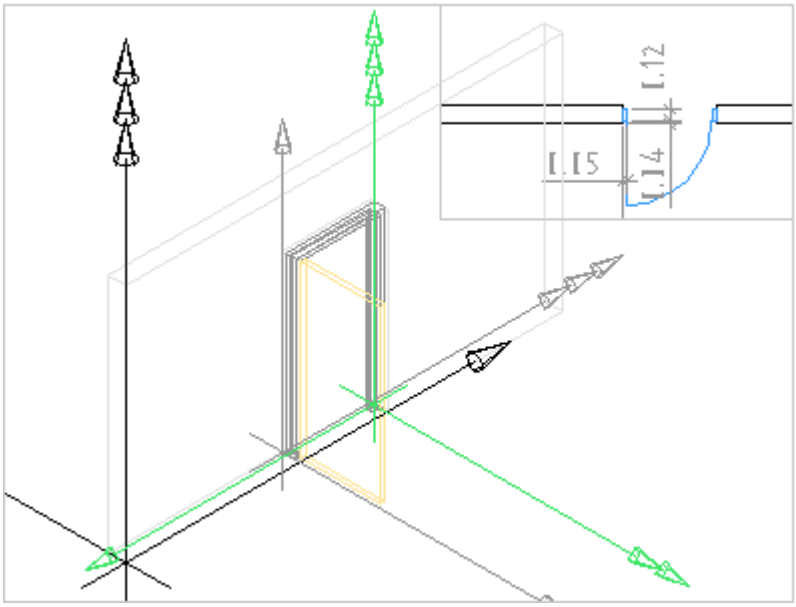


Figure 54 : single swing door (inward, right side hinge)

```
/* door definition */
#91=IFCDOOR('3cCAHy3JbB3eDyXfruy7HK',#16,$,'',$,#90,$,$,2.,1.);
#90=IFCLOCALPLACEMENT(#80,#89);
#89=IFCAXIS2PLACEMENT3D(#86,#87,#88);
#86=IFCCARTESIANPOINT((0.,0.04,1.));
#87=IFCDIRECTION((1.,0.,0.));
#88=IFCDIRECTION((0.,0.,-1.));
#92=IFCRELFILLSELEMENT('19RQHSNCP0shWm8ynf19RK',#16,$,$,#81,#91);

/* door type definitions */
#93=IFCRELDEFINESBYTYPE('3w12FoWQH2X9TTevy830PA',#16,$,$,(#91),#85);
#85=IFCDOORSTYLE('0Cq4HglY1DeAYBj0RB2XmO',#16,'Standard',$,'',(#83,#84),$,'',.SINGLE_SWI
NG_RIGHT.,.NOTDEFINED.,.T.,.F.);
#83=IFCDOORLININGPROPERTIES('3efP6cLufD_8xUjPp5u36X',#16,$,$,0.12,0.05,$,$,$,$,$,$,$,$
);
#84=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenxUhv',#16,$,$,0.05,.SWINGING.,1.,.LEFT.,$)
;
```

Other door types are inserted the same way, like double swing, sliding, folding, or revolving doors. The symbolic representation of those doors depends on the receiving system, not the symbolic representation is exchanged, but the definition parameter.

If the door type has two panels, two instances of *IfcDoorPanelProperties* needs to be inserted, the *PanelPosition* attribute determines which panel it is.

The following double swing door with unequal panel width is exchanged. The width of each of the two panels is given as a ratio measure.

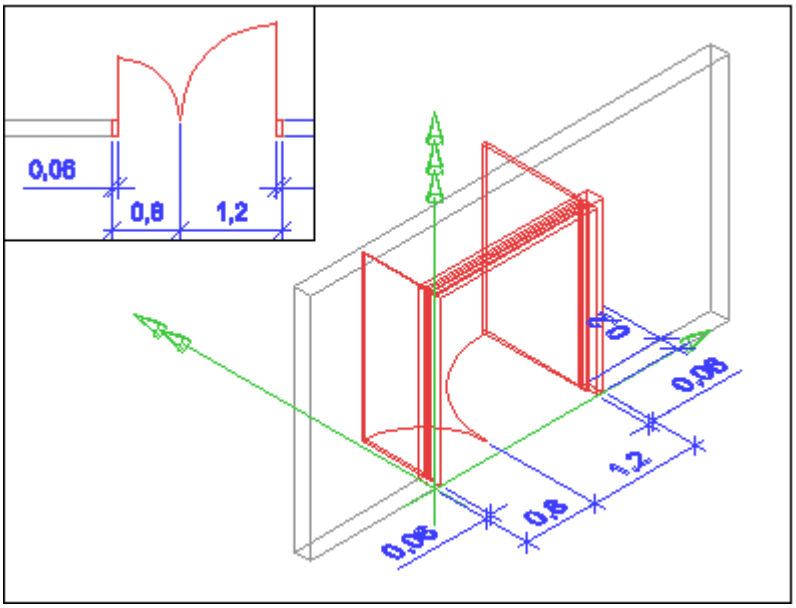


Figure 55 : Double swing door

```
/* door definition */
#91=IFCDOOR('3cCAHy3JbB3eDyXfruy7HK',#16,$,'',$,#90,$,$,2.,2.);
#90=IFCLOCALPLACEMENT(#80,#89);
#89=IFCAXIS2PLACEMENT3D(#86,#87,#88);
#86=IFCCARTESIANPOINT((0.,0.2,0.));
#87=IFCDIRECTION((1.,0.,0.));
#88=IFCDIRECTION((0.,0.,1.));
#92=IFCRELFILLSELEMENT('19RQHSNCP0shWm8ynf19RK',#16,$,$,#81,#91);

/* door type definitions */
#93=IFCRELDEFINESBYTYPE('3w12FoWQH2X9TTevy830PA',#16,$,$,(#91),#85);
#85=IFCDOORSTYLE('0Cq4HglY1DeAYBj0RB2XmO',#16,'Double swing door','Double swing door
with unequal door panels','',(#83,#84,#85),$,'', DOUBLE_DOOR_SINGLE_SWING,.NOTDEFINED.,
.T.,.F.);
#83=IFCDOORLININGPROPERTIES('3efP6cLufD_8xUjPp5u36X',#16,$,$,0.20,0.06,$,$,$,$,$,$,$,$
);
#84=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenxUhv',#16,$,$,0.03,.SWINGING.,0.4,.LEFT.,$
);
#85=IFCDOORPANELPROPERTIES('0$Z5by3D9BWxrscSenedgh',#16,$,$,0.03,.SWINGING.,0.6,.RIGHT.,
$);
```

5.4.2 Windows

Windows are defined for the IFC2x data sharing as individual instances of *IfcWindow* on the occurrence side and shared instances of *IfcWindowStyle*, including one instance of *IfcWindowLiningProperties* and at least one instance of *IfcWindowPanelProperties* at the type side. The conventions for the local placement (location and axis direction) and for the use of the style, lining and panel properties, follow the same principles as for the doors.

by *IfcWindow*

- The position of the window relative to the opening and wall (particularly the offset position to the wall edge),
- The opening direction of the window (swing in, swing out – this might not be shown by the symbolic representation of the receiving application, but is important for procurement)
- The overall width and height of the window
- The individual geometric (B-rep) representation (if given), as a mapped item (transformation)

by *IfcWindowStyle*

- The name and description of the style
- The operation type of the window (this includes also the distinction of left vs. right opening, as these are two different types from a manufacturing point of view)
- The construction type (for information purposes only)
- and two indications of the use of the (potentially) attached shared geometric (B-rep) representation (whether it overrides eventually given additional parameters and whether the attached geometry is sizable)

by *IfcWindowLiningProperties*

- (as minimum) depth and thickness
- (additional) mullions and transoms definitions, if the window style has more then one panel,

by *IfcWindowPanelProperties*

- (as minimum) depth and width of the panel
- operation type and position of the panel (the latter is important, if a multi panel window is exchanged, in this case two or three (depending on the number of panels) instances of *IfcWindowPanelProperties* are attached to the instance of *IfcWindowStyle*.

Note: The use of IfcWindowStyle is mandatory for each occurrence of IfcWindow. Therefore each instance of IfcWindow has to have a link IfcWindow ← IsDefinedBy – IfcRelDefinesByType – RelatingType → IfcWindowStyle. The provision of window lining and panel properties, i.e. IfcWindowLiningProperties and IfcWindowPanelProperties is optional (if not otherwise requested by a view definition), if a given property is not given in the IFC data, it can be replaced by a default value at the receiving system.

As for doors there are the same strict conventions on the use of the local placement for *IfcWindow*.

Location:

- the local origin of the window placement shall be the insertion point of the window lining (all length parameter of the lining and panel are positive and shall be applied along the positive side of the y-axis.

Axis direction:

- the x-axis shall point into the direction of the panel (when closed), this is parallel to the wall path or the tangent for curved walls in the middle of the opening,
- the y-axis shall point into the direction of the window opening
- the z-axis shall point vertical

The general use of the style information, by *IfcWindowStyle*, and the property set information for the window lining, by *IfcWindowLiningProperties*, and for the panels, by *IfcWindowPanelProperties*, follows the same conventions as provided for door.

The following example shows a standard single swing (or casement) window, which is opening to the left. The lining depth is 0.10m (within a 0.2m thick wall) and it is positioned centric within the wall. The lining thickness is 0.05. The overall height is 1.26m and the width is 1.01m. The sill height is 1.0m.

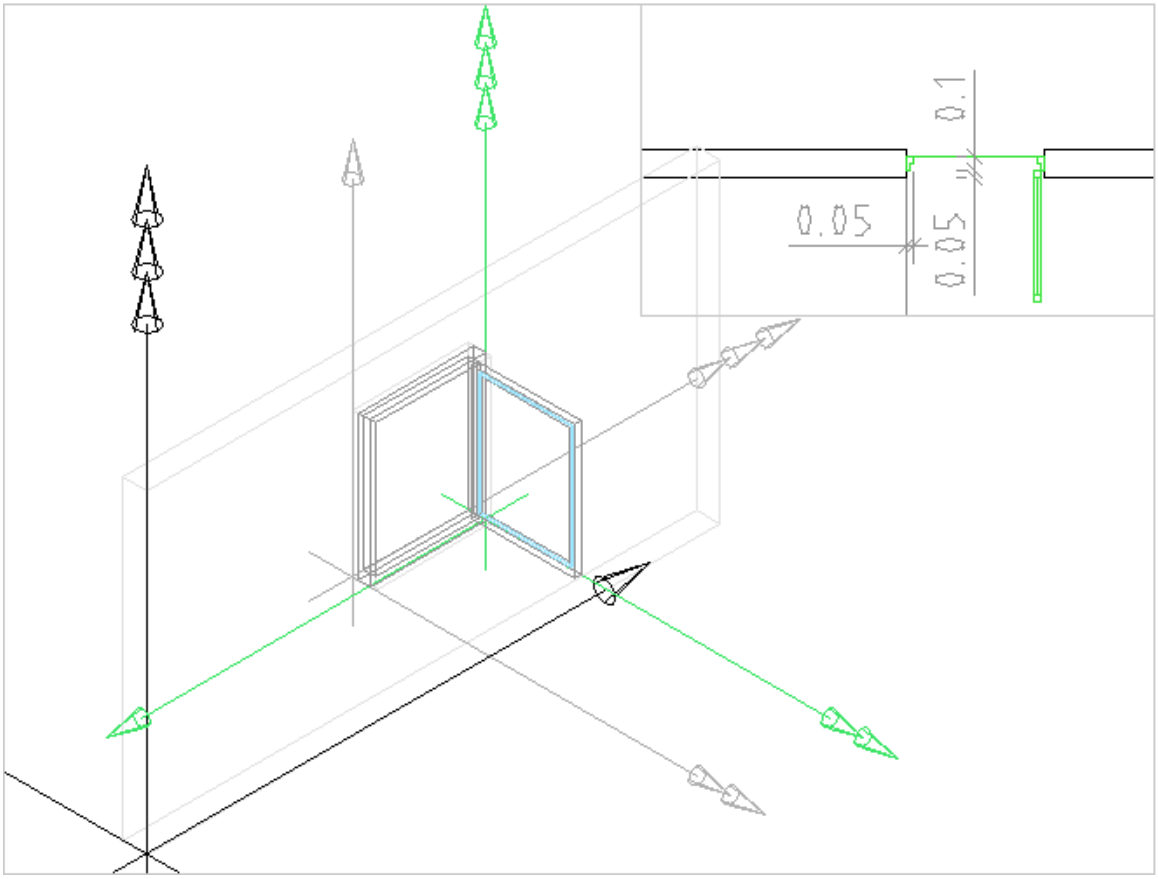


Figure 56 : Example of a single swing window

```
/* Window occurrence definitions */
#97=IFCRELFILLSELEMENT('3Y_rdf1p1FGAUa0WqR2Pgj',#16,$,$,#86,#96);
#98=IFCRELDEFINESBYTYPE('0E8SQc2rP8EuP0gwAozIDf',#16,$,$,(#96),#90);
#96=IFCWINDOW('1UTrMzo0b5V8DfGgSLiFqG',#16,$,'',$,#95,$,$,1.26,1.01);
#95=IFCLOCALPLACEMENT(#85,#94);
#94=IFCAXIS2PLACEMENT3D(#91,#92,#93);
#91=IFCCARTESIANPOINT((0.,0.05,1.01));
#92=IFCDIRECTION((1.,0.,0.));
#93=IFCDIRECTION((0.,0.,-1.));

/* Window style definitions */
#90=IFCWINDOWSTYLE('1bpFhGFobCm8tK$vXFs5_G',#16,$,$,'',(#88,#89),$,'',.NOTDEFINED.,.SING
LE_PANEL.,.T.,.F.);
#88=IFCWINDOWLININGPROPERTIES('19Rb5J55vBkuSUWRETAbBc',#16,$,$,0.1,0.05,$,$,$,$,$,$,$);
#89=IFCWINDOWPANELPROPERTIES('2BouMUXer8Dgf7PkPulxj3',#16,$,$,.SIDEHUNGLEFTHAND.,.NOTDEF
INED.,0.05,0.05,$);

/* Opening definitions */
#87=IFCRELVOIDSELEMENT('3a9mqKUWD7Guk3JRSUc788',#16,$,$,#61,#86);
#86=IFCOPENINGELEMENT('0pg1ZN4LTBmfIOJAerUV76',#16,$,'',$,#85,#80,$);
#85=IFCLOCALPLACEMENT(#36,#84);
#84=IFCAXIS2PLACEMENT3D(#81,#82,#83);
#81=IFCCARTESIANPOINT((2.,0.2,1.));
#82=IFCDIRECTION((1.,0.,0.));
#83=IFCDIRECTION((0.,0.,1.));
#80=IFCPRODUCTDEFINITIONSHAPE($,$,(#79));
#79=IFCSHAPEREPRESENTATION(#11,$,'SweptSolid',(#77));
#77=IFCEXTRUDEDAREASOLID(#70,#75,#76,0.2);
#70=IFCRECTANGLEPROFILEDEF(.AREA.,$,#69,1.26,1.01);
#69=IFCAXIS2PLACEMENT2D(#67,#68);
#67=IFCCARTESIANPOINT((0.,0.));
#68=IFCDIRECTION((1.,0.));
#75=IFCAXIS2PLACEMENT3D(#72,#73,#74);
#72=IFCCARTESIANPOINT((0.63,0.,0.505));
#73=IFCDIRECTION((0.,1.,0.));
#74=IFCDIRECTION((1.,0.,2.220446049250313E-016));
#76=IFCDIRECTION((0.,0.,1.));
```

5.5 Slabs

Slabs are exchanged by an IFC2x file as instances of *IfcSlab*. The concept of slab comprises all building elements that are usually planar and non-vertical, such as:

- floor slabs
- roof slabs
- stair landings

These types are differentiated by using the attribute *PredefinedType*, which datatype is an enumeration allowing the selection of a predefined enumerators or the use of additional user defined types. The use of *PredefinedType* is required, if no type object through *IfcSlabType* is assigned using the *IsDefinedBy* inverse relationship.

```
TYPE IfcSlabTypeEnum = ENUMERATION OF
  (FLOOR, ROOF, LANDING, USERDEFINED, NOTDEFINED);
END_TYPE;
```

Standard forms of slabs are determined by the shape representation of the slab and include all cases of geometry under the following conditions:

- having a single (material and extrusion) thickness,
- are extruded perpendicular to the profile,
- are not clipped at any end.

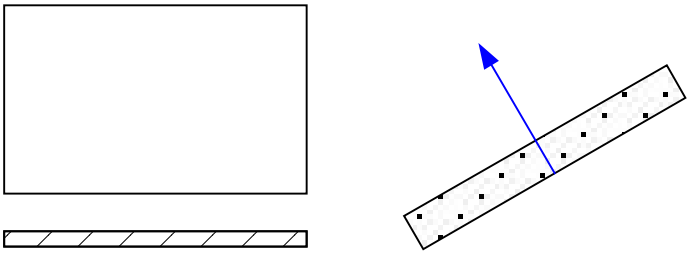


Figure 57 : Examples for standard slabs

Standard slabs may have openings and have material information assigned. Material information for standard slabs is given by material layers, using the *IfcMaterialLayerSetUsage* entity. For information on material layers see section 10.2.1.3.

Special forms of slabs are determined by a more complex shape representation which includes the following:

- having a varying thickness or special construction types
- are extruded non-perpendicular to the profile
- are clipped at some ends

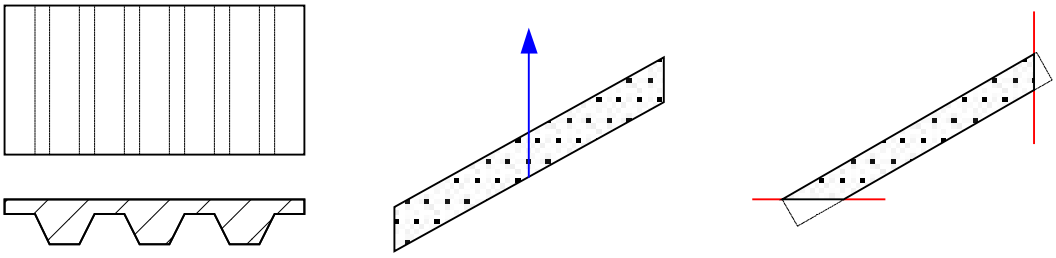


Figure 58 : Examples for special slabs

Special slabs may have openings and have material information assigned. Material information for special slabs may be given by material layers or (if there is no layering structure) by a material list. For information on material lists see section 10.2.1.2.

The following units of functionality (UoF) are common for both types of slabs definitions, they are already discussed earlier, as they apply to a higher level in the hierarchy.

- placement (→ 9.1.2)
- element in space containment
- element quantities (→ 10.3.1)
- element space boundaries (→ 4.2.4.1)
- element opening voiding (→ 5.3.4)

5.5.1 Geometric representations of slabs

The UoF of multiple shape representation of slabs includes (at least) one shape representations for each instance of *IfcSlab*. It represents the slab body. The body is provided by the following geometric representation items

- *IfcExtrudedAreaSolid* (for all standard slabs)
- *IfcBooleanClippingResult* (for special slabs with edge cuttings)[13]
- *IfcFacetedBrep* (for special walls with complex geometry)

In all cases the value for *IfcShapeRepresentation.RepresentationIdentifier* is "Body". If a slab has openings or recesses and trenches, they are exchanged using *IfcOpeningElement*, assigned to the slab through the *IfcRelVoidsElement* and accessibly from the *IfcSlab* entity through the *HasOpenings* inverse relationship. For more information on openings see section 5.3 and for slab opening see section 5.3.4.

NOTE: In the IFC model, a slab has to have a single body, it is not allowed (according to the current view definitions) to export a slab having two totally separated bodies (either as swept solids, or as BREP). Therefore the sending system has to split the slab into two (or more) slabs before exporting it to IFC, if it handles slabs with two (or more) separate bodies.

5.5.1.1 Geometric representations of standard slabs

The slab body is given by an instance of *IfcShapeRepresentation* with the following conventions:

- use of *IfcShapeRepresentation.RepresentationIdentifier* = "Body"
- use of *IfcShapeRepresentation.RepresentationType* = "SweptSolid"
- use of *IfcExtrudedAreaSolid* (with *IfcArbitraryClosedProfileDef* as the underlying profile definition) for *IfcShapeRepresentation.Items[1].*

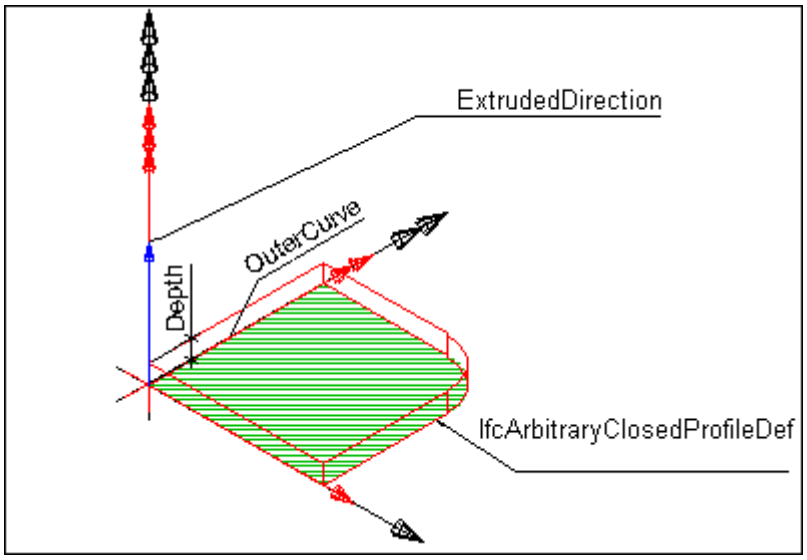


Figure 59 : Example for the shape representation of a standard slab

Figure 58 shows the shape representation of a standard slab. The slab profile (its foot print area) is given by an *IfcArbitraryClosedProfileDef*. The *OuterCurve* attribute references an *IfcCompositeCurve* (in this case due to the round edges) or by an *IfcPolyline* (for polygonal boundaries) or by another

[13] Note that within the current IFC2x coordination view clippings by using the *IfcBooleanClippingResult* are not supported, such shapes need to be exchanged using the *IfcFacetedBrep*.

subtype of *IfcBoundedCurve* (in general). The *IfcExtrudedAreaSolid.Depth* attribute represents the thickness of the slab.

> *NOTE: If the material information is attached using the IfcMaterialLayerSetUsage, then the IfcMaterialLayerSet.TotalThickness attribute shall have the same value as the attribute IfcExtrudedAreaSolid.Depth. This needs to be guaranteed by the sending application.*

The value of the *ExtrusionDirection* for standard slabs needs to be perpendicular to the XY plane of the *IfcExtrudedAreaSolid.Position*.

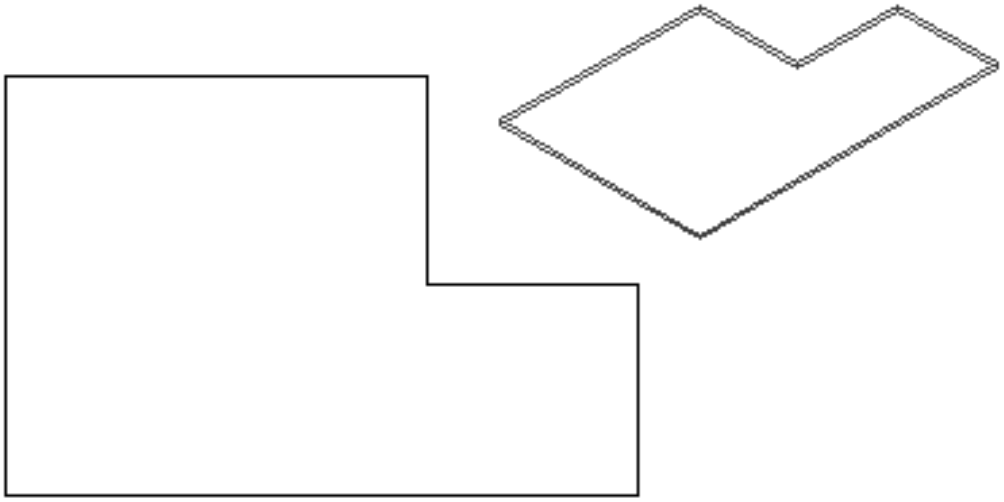


Figure 60 : Simple horizontal slab example

Example:

> *The simple floor slab has a polygonal footprint, and a thickness of 0.2 meter. The following ifc file shows its geometric representation.*

```
#47=IFCSLAB('1I6mYbG99ESgWwGNEv5i31',#16,$,'',$,#28,#46,$,.FLOOR.);
 #28=IFCLOCALPLACEMENT(#22,#27);
  /* local placement data of building story skipped */
  #27=IFCAXIS2PLACEMENT3D(#24,#25,#26);
   #24=IFCCARTESIANPOINT((5.,5.,0.));
   #25=IFCDIRECTION((0.,0.,1.));
   #26=IFCDIRECTION((1.,0.,0.));
 #46=IFCPRODUCTDEFINITIONSHAPE($,$,(#45));
  #45=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#43));
   #43=IFCEXTRUDEDAREASOLID(#37,#41,#42,0.2);
    #37=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#36);
     #36=IFCPOLYLINE((#29,#30,#31,#32,#33,#34,#35));
      #29=IFCCARTESIANPOINT((0.,0.));
      #30=IFCCARTESIANPOINT((15.,0.));
      #31=IFCCARTESIANPOINT((15.,5.));
      #32=IFCCARTESIANPOINT((10.,5.));
      #33=IFCCARTESIANPOINT((10.,10.));
      #34=IFCCARTESIANPOINT((0.,10.));
      #35=IFCCARTESIANPOINT((0.,0.));
    #41=IFCAXIS2PLACEMENT3D(#38,#39,#40);
     #38=IFCCARTESIANPOINT((0.,0.,0.));
     #39=IFCDIRECTION((0.,0.,1.));
     #40=IFCDIRECTION((1.,0.,0.));
    #42=IFCDIRECTION((0.,0.,1.));
```

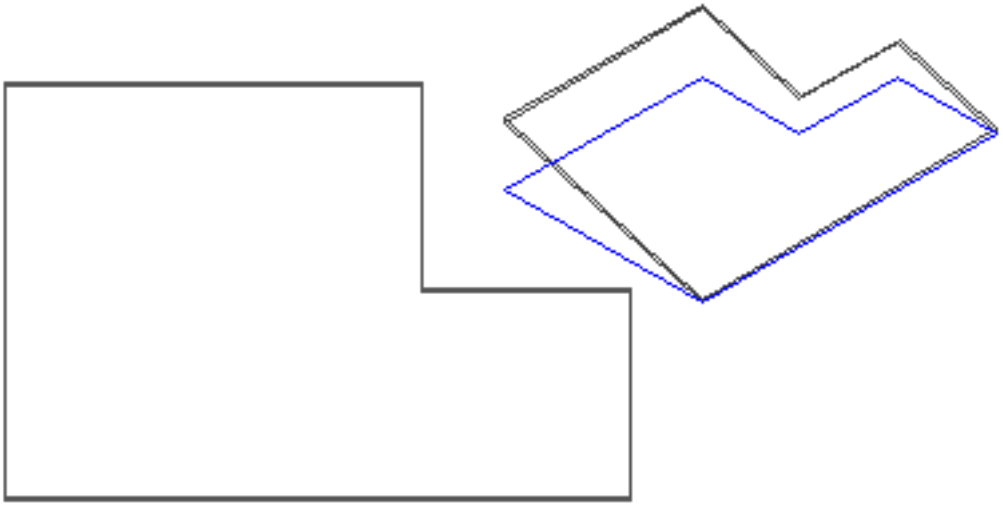


Figure 61 : Simple sloped slab example

Example

The same simple floor slab, this time with a slope of the whole body (but still perpendicular extrusion), the slope is 30%, or approximately 16,7'. The blue line is for visualization purposes only and not exchanged.

```
#47=IFCSLAB('0TuXShKmL97xliSYeLoNK0',#16,$,'',$,#28,#46,$,.FLOOR.);
 #28=IFCLOCALPLACEMENT(#22,#27);
  /* local placement data of building story skipped */
  #27=IFCAXIS2PLACEMENT3D(#24,#25,#26);
   #24=IFCCARTESIANPOINT((5.,5.,0.));
   #25=IFCDIRECTION((0.,-0.2873478855663453,0.9578262852211514));
   #26=IFCDIRECTION((1.,0.,0.));
 #46=IFCPRODUCTDEFINITIONSHAPE($,$,(#45));
  #45=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#43));
   #43=IFCEXTRUDEDAREASOLID(#37,#41,#42,0.2);
    #37=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#36);
     #36=IFCPOLYLINE((#29,#30,#31,#32,#33,#34,#35));
      #29=IFCCARTESIANPOINT((0.,0.));
      #30=IFCCARTESIANPOINT((15.,0.));
      #31=IFCCARTESIANPOINT((15.,5.220153254455275));
      #32=IFCCARTESIANPOINT((10.,5.220153254455275));
      #33=IFCCARTESIANPOINT((10.,10.44030650891055));
      #34=IFCCARTESIANPOINT((0.,10.44030650891055));
      #35=IFCCARTESIANPOINT((0.,0.));
    #41=IFCAXIS2PLACEMENT3D(#38,#39,#40);
     #38=IFCCARTESIANPOINT((0.,0.,0.));
     #39=IFCDIRECTION((0.,0.,1.));
     #40=IFCDIRECTION((1.,0.,0.));
    #42=IFCDIRECTION((0.,0.,1.));
```

6 Building services elements and related concepts

The support for building service elements has been greatly enhanced in IFC2x2, the use of types (as subtypes of *IfcTypeProduct*) is now strongly encouraged. Also the definitions for ports (as *IfcDistributionPort*) have been reworked.

6.1 Building services type, occurrence and performance history entities

In IFC2x2, the notions of type and occurrence have been formalized in the building services model architecture, as has the concept of a performance history entity which is used to store state-specific properties that come and go during the life-cycle of the occurrence. The occurrence entity is related to its type via the *IfcRelDefinesByType* relationship and to zero or many *IfcPerformanceHistory* entities via the *IfcRelAssignsToControl* relationship. The following figure illustrates these concepts:

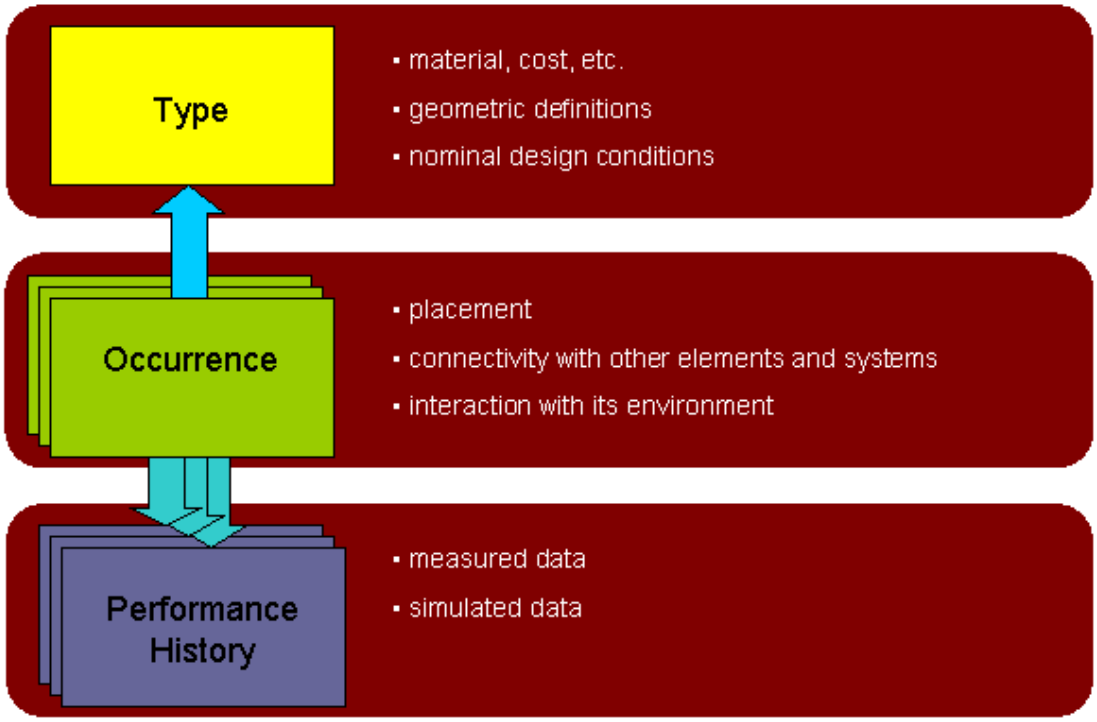


Figure 62 : Type, Occurrence and Performance History Concepts

Each type specific entity has an attribute *PredefinedType*. It refer to an enumeration of special types of the component, which further specializes the component tree and allows the allocation of specific property sets to the type.

The depth of the subtyping tree is different for the occurrences and for the types, the occurrence specialization tree is shallow, whereas the type specialization tree is deep. The listing of all occurrence entities with the applicable type entities and the predefined enumeration is shown in Table 8. within Appendix 1.

6.1.1 Concepts of building service type definitions

Building service type definitions are used to define common properties for occurrence elements. Each instance of a building service occurrence element gets its full description (aside from the local placement, local connectivity, occurrence properties, etc.) from its type definition including the block-type geometry, shared properties, basic type name and description, etc. These are exchanged through IFC2x2 as subtypes of *IfcTypeProduct*.

Therefore each building service element subtype of *IfcFlowDistributionElement* refers to an instance of an *IfcTypeProduct* subtype using the *IfcRelDefinesByType* relationship. The *IfcTypeProduct* handles the:

- type name as *IfcTypeProduct.Name*
- description of the type as *IfcTypeProduct.Description*
- shared properties, common for all occurrences, as *IfcTypeProduct.HasPropertySets*
- shared geometric representation as *IfcTypeProduct.RepresentationMaps*

This information is independent of the actually inserted building service element and may even be exchanged without any occurrence instance. In this case it enables the exchange of catalogues or libraries of type definitions.

Within IFC2x2 the building service type definition has been greatly enhanced. For each entity representing an occurrence of a building service component, an equivalent abstract type object exists. All inherit from the *IfcDistributionElementType*.

```
ENTITY IfcDistributionElementType;
   GlobalId             :     IfcGloballyUniqueId;
   OwnerHistory         :     IfcOwnerHistory;
   Name                 :     OPTIONAL IfcLabel;
   Description          :     OPTIONAL IfcText;
   ApplicableOccurrence :     OPTIONAL IfcLabel;
   HasPropertySets      :     OPTIONAL LIST [1:?] OF UNIQUE IfcPropertySetDefinition;
   RepresentationMaps   :     OPTIONAL LIST [1:?] OF UNIQUE IfcRepresentationMap;
   Tag                  :     OPTIONAL IfcLabel;
   ElementType          :     OPTIONAL IfcLabel;
INVERSE
   HasAssociations      :     SET OF IfcRelAssociates FOR RelatedObjects;
   ObjectTypeOf         :     SET [0:1] OF IfcRelDefinesByType FOR RelatingType;
END_ENTITY;
```

Each non abstract subtype of *IfcDistributionElementType* adds another attribute, the *PredefinedType*. The *PredefinedType* represents a further level of specialization. It can either be set to a specific enumerator or to either .*USERDEFINED*. or .*NOTDEFINED*.. In case of .*USERDEFINED*. the attribute *ElementType* should hold a user specific component type name, in case of .*NOTDEFINED*. no further specialization is available.

6.1.1.1 Representation maps

The representation maps specify one or many representations of the building service element (and they are therefore comparable to the concept of multi-view blocks or library objects, or intelligent macros, as often used in CAD systems). Each representation map has an origin (normally 0.,0.,0.) and a representation, which has a representation identifier and representation type, identifying its use within different representation views (see also 9.1.4.1).

Example: A specific elbow type (here IfcDuctFittingType) should be exchanged as the underlying type of (one or several) building service elements (here IfcFlowFitting). The example shows how are the name and description is exchanged. It also shows, how two different representation maps (one for the body used in the 3D model view, the second for the footprint used in the floor plan view) can be sent.

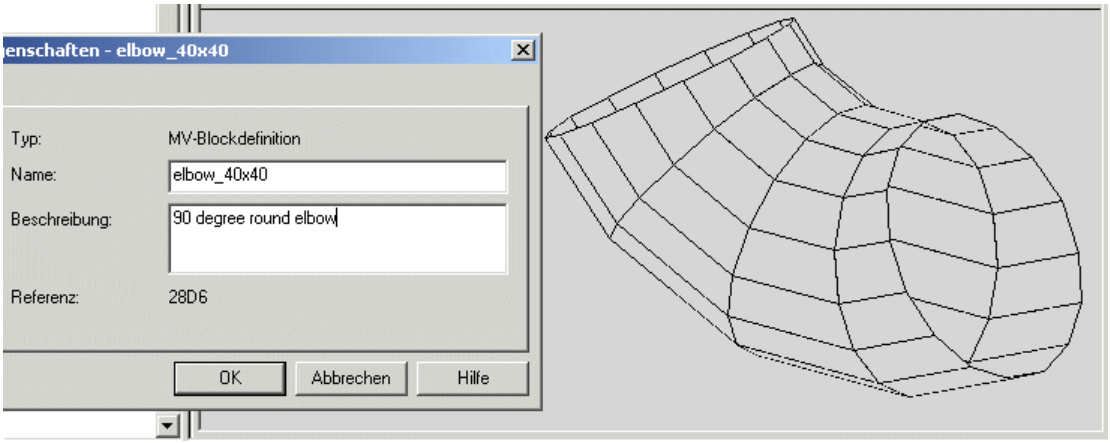


Figure 63 : example of block definition for elbow

```
/* Type definition of 40x40 elbow */
#1=IFCDUCTFITTINGTYPE('abcdefghijklmnopqrst12', #2, 'elbow_40x40', '90 degree round
elbow', 'IfcFlowFitting', $, (#10,#11), $, $, .BEND.);

/* first representation for floor plan view */
#10=IFCREPRESENTATIONMAP(#12, #13);
#12=IFCAXIS2PLACEMENT3D(#14, $, $);
#14=IFCCARTESIANPOINT((0.,0.,0.));
#13=IFCSHAPEREPRESENTATION(#20, 'FootPrint', 'GeometricCurveSet', (#15));
#15=IFCCOMPOSITECURVE(#101, .F.);
#20=IFCGEOMETRICREPRESENTATIONCONTEXT('FloorPlan', 'Design', 3, $, #22, $);

/* second representation for model view */
#11=IFCREPRESENTATIONMAP(#16, #17);
#16=IFCAXIS2PLACEMENT3D(#18, $, $);
#18=IFCCARTESIANPOINT((0.,0.,0.));
#17=IFCSHAPEREPRESENTATION(#21, 'Body', 'SurfaceModel', (#19));
#19=IFCSHELLBASEDSURFACEMODEL($);
#21=IFCGEOMETRICREPRESENTATIONCONTEXT('Model', 'Design', 3, $, #22, $);
```

6.1.1.2 Representation types

There are several geometric representations foreseen for building service types:

- Representation identifier 'Axis', representation type 'GeometricSet'
- Representation identifier 'FootPrint', representation type 'GeometricSet'
- Representation identifier 'Body', representation type 'SurfaceModel', 'Brep' or 'SweptSolid'

Each of the representation types can be replaced by "MappedRepresentation", if the type concept, i.e. the use of an *IfcMappedItem* for sharing the geometric representation(s) of the type is used. The principle concept of using *IfcMappedItem* is introduced in 9.1.4.2, its use in conjunction with the *IfcTypeProduct* is shown in 10.1.4.

From IFC2x2 onwards the use of the type concept is now strongly encouraged, the entity type information about the building service element now lies in the type object.

Example: The following example shows four wash basins (as a type of IfcSanitaryTerminalType and occurrences of IfcFlowTerminal) inserted as instances of a library object. Each instance has its own local placement, and the geometric representation of its 'Body' is given by 'MappedRepresentation', being the IfcMappedItem, referring to the common IfcRepresentationMap. Each mapped item has a transformation matrix relative to the object coordinate system, given by the IfcLocalPlacement, (here always scale 1 and no translation or rotation).

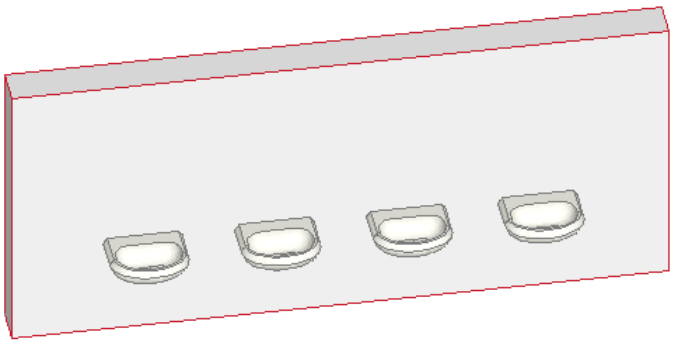


Figure 64 : Example of building service element occurrences

```
/* wash basin type and block geometry */
#1=IFCSANITARYTERMINALTYPE('abcdefghijklmnopqrst11', #2, 'Waschbecken_1 70', $,
'IfcFlowTerminal', (#10), (#1011), $, $, .URINAL.);
#1011=IFCREPRESENTATIONMAP(#1016, #1017);
#1016=IFCAXIS2PLACEMENT3D(#1018, $, $);
#1017=IFCSHAPEREPRESENTATION(#1021, 'Body', 'SurfaceModel', (#1019));
#1018=IFCCARTESIANPOINT((0.,0.,0.));
#1019=IFCSHELLBASEDSURFACEMODEL(/* content omitted */);
#1021=IFCGEOMETRICREPRESENTATIONCONTEXT('Model', 'Design', 3, $, #1022, $);
#1022=IFCAXIS2PLACEMENT3D(#1023, $, $);
```

```
#1023=IFCCARTESIANPOINT((0.,0.,0.));

/* assignment of instances to the common type */
#1000=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst17', #2, $, $, (#17,#15,#14,#16), #1);

/* first wash basin */
#14=IFCFLOWTERMINAL('abcdefghijklmnopqrst13', #2, $, $, $, #101, #102, $);
#101=IFCLOCALPLACEMENT($, #104);
#102=IFCPRODUCTDEFINITIONSHAPE($, $, (#103));
#103=IFCSHAPEREPRESENTATION(#1021, 'Body', 'MappedRepresentation', (#1024));
#104=IFCAXIS2PLACEMENT3D(#405, $, $);
#105=IFCCARTESIANPOINT((1.,5.,0.8));
#1024=IFCMAPPEDITEM(#1011, #1025);
#1025=IFCCARTESIANTRANSFORMATIONOPERATOR3D($, $, #1026, $, #1027);
#1026=IFCCARTESIANPOINT((0.,0.,0.));
#1027=IFCDIRECTION((0.,0.,1.));

/* second wash basin */
#15=IFCFLOWTERMINAL('abcdefghijklmnopqrst14', #2, $, $, $, #201, #202, $);
#201=IFCLOCALPLACEMENT($, #204);
#202=IFCPRODUCTDEFINITIONSHAPE($, $, (#203));
#203=IFCSHAPEREPRESENTATION(#1021, 'Body', 'MappedRepresentation', (#1124));
#204=IFCAXIS2PLACEMENT3D(#405, $, $);
#205=IFCCARTESIANPOINT((2.5,5.,0.8));
#1124=IFCMAPPEDITEM(#1011, #1125);
#1125=IFCCARTESIANTRANSFORMATIONOPERATOR3D($, $, #1126, $, #1127);
#1126=IFCCARTESIANPOINT((0.,0.,0.));
#1127=IFCDIRECTION((0.,0.,1.));

/* third wash basin */
#16=IFCFLOWTERMINAL('abcdefghijklmnopqrst15', #2, $, $, $, #301, #302, $);
#301=IFCLOCALPLACEMENT($, #304);
#302=IFCPRODUCTDEFINITIONSHAPE($, $, (#303));
#303=IFCSHAPEREPRESENTATION(#1021, 'Body', 'MappedRepresentation', (#1224));
#304=IFCAXIS2PLACEMENT3D(#405, $, $);
#305=IFCCARTESIANPOINT((4.,5.,0.8));
#1224=IFCMAPPEDITEM(#1011, #1225);
#1225=IFCCARTESIANTRANSFORMATIONOPERATOR3D($, $, #1226, $, #1227);
#1226=IFCCARTESIANPOINT((0.,0.,0.));
#1227=IFCDIRECTION((0.,0.,1.));

/* fourth wash basin */
#17=IFCFLOWTERMINAL('abcdefghijklmnopqrst16', #2, $, $, $, #401, #402, $);
#401=IFCLOCALPLACEMENT($, #404);
#402=IFCPRODUCTDEFINITIONSHAPE($, $, (#403));
#403=IFCSHAPEREPRESENTATION(#1021, 'Body', 'MappedRepresentation', (#1324));
#404=IFCAXIS2PLACEMENT3D(#405, $, $);
#405=IFCCARTESIANPOINT((5.5,5.,0.8));
#1324=IFCMAPPEDITEM(#1011, #1325);
#1325=IFCCARTESIANTRANSFORMATIONOPERATOR3D($, $, #1326, $, #1327);
#1326=IFCCARTESIANPOINT((0.,0.,0.));
#1327=IFCDIRECTION((0.,0.,1.));
```

The examples show how the geometry that is associated to the *IfcTypeProduct* subtype *IfcSanitaryTerminalType* is referenced by the *IfcMappedItem* (which could be done for multiple geometric representations). The Cartesian transformation operator places the *IfcRepresentationMap* (given within its own coordinate system, provided by *IfcRepresentationMap.MappingOrigin*) into the object coordinate system (given by supertype *IfcProduct.ObjectPlacement*). Although not used in the example, translation, rotation and scaling (uniform and non-uniform) of the *IfcRepresentationMap* is possible.

6.1.1.3 Assigning properties to the type definition

In addition to the shared geometric representation each building service element type can also hold the shared (or type specific) properties (including property values). These are exchanged as property sets directly attached to the type (and do not need to be multiplied for each occurrence) using the *IfcTypeObject.HasPropertySets* attribute.

> *Example: A washbasin type (here IfcSanitaryTerminalType) should be exchanged as the underlying type of (one or several) building service elements (here IfcFlowTerminal). The example shows how are the name and description is exchanged. It also shows, how the commonly shared properties are exchanged (the geometric representation is omitted in that example).*

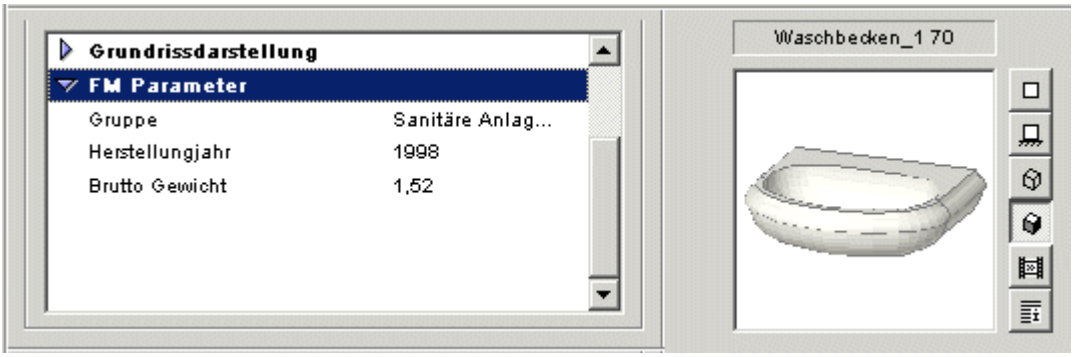


Figure 65 : example of block or library definition for washbasin

```
/* Type definition of wash basin */
#1=IFCSANITARYTERMINALTYPE('abcdefghijklmnopqrst12', #2, 'Waschbecken_1 70', $,
'IfcFlowTerminal', (#10), $, $, $, .URINAL.);

/* shared property definitions for the type */
#10=IFCPROPERTYSET('abcdefghijklmnopqrst13', #2, 'FM Parameter', $, (#12,#13,#11));
#11=IFCPROPERTYSINGLEVALUE('Group', $, IFCLABEL('Sanit\S\dre Anlagen'), $);
#12=IFCPROPERTYSINGLEVALUE('Production Year', $, IFCNUMERICMEASURE(1998), $);
#13=IFCPROPERTYSINGLEVALUE('Gross Weight', $, IFCNUMERICMEASURE(1.52), $);
```

6.1.2 Concepts of building service occurrence elements

The occurrence entity represents the physical manifestation of the type. All building service elements inherit the concepts and UoF associated to elements, which are provided by *IfcElement*. It includes:

- "Identification" and "Change management", see section 3.1.
- "Generic Type Assignment" and "Generic Property Attachment", see section 9.2, refined as element type assignment for selected building elements.
- "External References" to documents, classifications and libraries, see section 10.4.
- "Generic Decomposition"
- "Placement" see section 9.1.2, and "Shape Representation", see section 9.1.3, refined as building element specific shape representation (also referred to as geometry use cases) within this section.
- "Element in Spatial Structure Containment", see section 3.
- "Element quantities", see section 10.3.
- "Material assignment", see section 10.2. refined as building element specific material assignment (e.g. for walls) within this section.
- "Element Connectivity", see section 6.2.

All building service element occurrence entities are exchanged by an IFC2x2 file as instances of subtypes of *IfcDistributionElement*. If the appropriate semantic classification exists in the IFC2x2 specification (and within the originating application) then the building element is exported by the appropriate subtype entity. If this is not the case, then the *IfcDistributionElement* entity should be used instead.

> *NOTE: The use of IfcBuildingElementProxy is not encouraged and might be limited to the current edition of IFC2x2.*

Figure 65 shows the hierarchy for all currently defined building service occurrence elements. All subtypes of *IfcDistributionElement* are out of the stable platform.

> *NOTE: The area of building service elements will continue to undergo changes with future editions of the IFC2x2 model.*

The various subtypes of *IfcDistributionFlowElement* do not elaborate any further attributes. These subtypes are introduced solely for the purpose of semantics and logical structuring of the model.. Each of the subtypes inherit the following attributes and inverse relationships from *IfcDistribitionFlowElement*:

```
ENTITY IfcDistributionFlowElement
   GlobalId          : IfcGloballyUniqueId;
   OwnerHistory      : IfcOwnerHistory;
   Name              : OPTIONAL IfcLabel;
   Description       : OPTIONAL IfcText;
   ObjectType        : OPTIONAL IfcLabel;
   ObjectPlacement   : OPTIONAL IfcObjectPlacement;
   Representation    : OPTIONAL IfcProductRepresentation;
   Tag               : OPTIONAL IfcIdentifier;

INVERSE
   IsDefinedBy       : SET OF IfcRelDefines;
   HasAssociations   : SET OF IfcRelAssociates;
   HasAssignments    : SET OF IfcRelAssigns;
   Decomposes        : SET [0:1] OF IfcRelDecomposes;
   IsDecomposedBy    : SET OF IfcRelDecomposes;
   ReferencedBy      : SET OF IfcRelAssignsToProduct;
   ConnectedTo       : SET OF IfcRelConnectsElements;
   ConnectedFrom     : SET OF IfcRelConnectsElements;
   FillsVoids        : SET [0:1] OF IfcRelFillsElement;
   HasCoverings      : SET OF IfcRelCoversBldgElements;
   HasProjections    : SET OF IfcRelProjectsElement;
   HasOpenings       : SET OF IfcRelVoidsElement;
   HasPorts          : SET OF IfcRelConnectsPortToElement;
   IsConnectionRealization : SET OF IfcRelConnectsWithRealizingElements;
   ProvidesBoundaries      : SET OF IfcRelSpaceBoundary;
   ContainedInStructure    : SET [0:1] OF IfcRelContainedInSpatialStructure;
   HasControlElements      : SET [0:1] OF IfcRelFlowControlElements;
END_ENTITY;
```

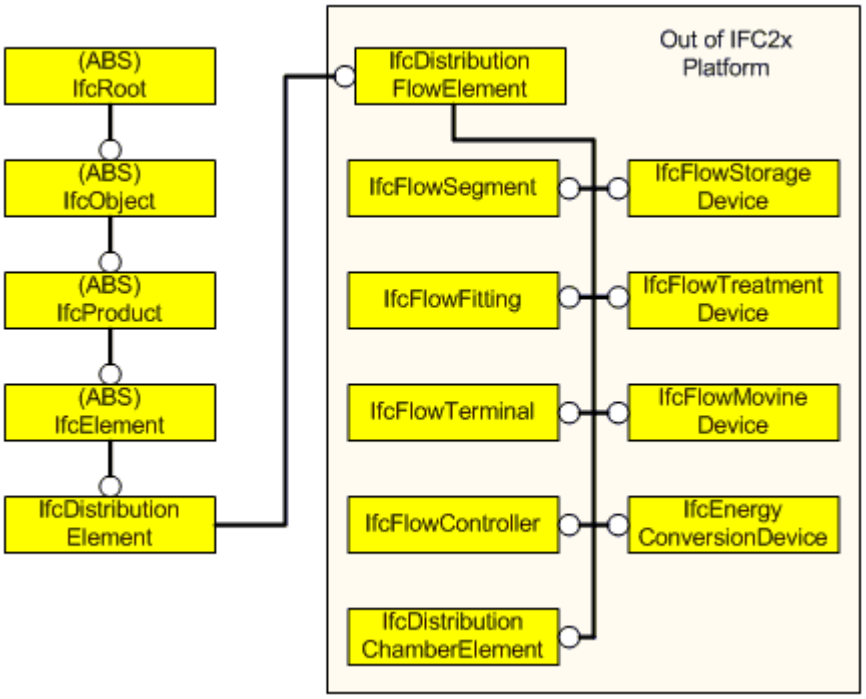


Figure 66 : Hierarchy chart of building service elements

6.1.2.1 Assigning properties to the occurrence element

Beside the shared properties already attached to the type as shown above, each occurrence of a building service element can have its occurrence specific properties attached, using the general principle as introduced in 10.1.2.1.

Example: Extending the example, reflected in Figure 63 by occurrence specific properties, two additional properties should be added: a serial number and an asset number. Both vary by occurrence for each of the four washbasins. The complete sum of properties for each occurrence of a building service element is therefore the sum of those property sets assigned thought the shared type (via IfcRelDefinesByType) and those assigned directly (through IfcRelAssignsByProperties).

```
/* definition of the type -no geometry shown- */
#1= IFCSANITARYTERMINALTYPE('abcdefghijklmnopqrst11', #2, 'Waschbecken_1 70', $,
'IfcFlowTerminal', (#10), (#1011), $, $, .URINAL.);
#10=IFCPROPERTYSET('abcdefghijklmnopqrst12', #2, 'FM Parameter', $, (#11,#12,#13));
#11=IFCPROPERTYSINGLEVALUE('Group', $, IFCLABEL('Sanit\S\dre Anlagen'), $);
#12=IFCPROPERTYSINGLEVALUE('Production year', $, IFCNUMERICMEASURE(1998), $);
#13=IFCPROPERTYSINGLEVALUE('Brutto Gewicht', $, IFCNUMERICMEASURE(1.52), $);

/* definition of the occurrences  -no geometry shown- */
#1000=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst17', #2, $, $, (#15,#16,#17,#14), #1);

#14=IFCFLOWTERMINAL('abcdefghijklmnopqrst13', #2, $, $, $, #101, #102, $);
#2028=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst23', #2, $, $, (#14), #2032);
#2032=IFCPROPERTYSET('abcdefghijklmnopqrst24', #2, 'FM occurrence parameter', $,
(#2034,#2033));
#2033=IFCPROPERTYSINGLEVALUE('Asset number', $, IFCLABEL('DE_A_234234'), $);
#2034=IFCPROPERTYSINGLEVALUE('Serial number', $, IFCLABEL('BD124-GF5835'), $);

#15=IFCFLOWTERMINAL('abcdefghijklmnopqrst14', #2, $, $, $, #201, #202, $);
#3028=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst25', #2, $, $, (#15), #3032);
#3032=IFCPROPERTYSET('abcdefghijklmnopqrst26', #2, 'FM occurrence parameter', $,
(#3034,#3033));
#3033=IFCPROPERTYSINGLEVALUE('Asset number', $, IFCLABEL('DE_A_234624'), $);
#3034=IFCPROPERTYSINGLEVALUE('Serial number', $, IFCLABEL('BD124-GF1534'), $);

#16=IFCFLOWTERMINAL('abcdefghijklmnopqrst15', #2, $, $, $, #301, #302, $);
#4028=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst27', #2, $, $, (#16), #4032);
#4032=IFCPROPERTYSET('abcdefghijklmnopqrst28', #2, 'FM occurrence parameter', $,
(#4034,#4033));
#4033=IFCPROPERTYSINGLEVALUE('Asset number', $, IFCLABEL('DE_A_234143'), $);
#4034=IFCPROPERTYSINGLEVALUE('Serial number', $, IFCLABEL('BD124-GF4294'), $);

#17=IFCFLOWTERMINAL('abcdefghijklmnopqrst16', #2, $, $, $, #401, #402, $);
#5028=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst32', #2, $, $, (#16), #5032);
#5032=IFCPROPERTYSET('abcdefghijklmnopqrst34', #2, 'FM occurrence parameter', $,
(#5034,#5033));
#5033=IFCPROPERTYSINGLEVALUE('Asset number', $, IFCLABEL('DE_A_234463'), $);
#5034=IFCPROPERTYSINGLEVALUE('Serial number', $, IFCLABEL('BD124-GF78253'), $);
```

6.1.2.2 Overriding properties assigned to the type in the occurrence element

In addition there is the possibility to override shared properties, assigned through the type by using the overriding mechanism of *IfcRelOverridesProperties*. This allows the ability to handle exceptions for a few shared properties that have different values for some occurrences. The principle mechanism is described in 10.1.2.2

Example: One occurrence of the washbasins has a different production year (although otherwise identical), this can be exchanged by overriding the shared property, attached to the type.

```
/* definition of the type -no geometry shown- */
#1= IFCSANITARYTERMINALTYPE('abcdefghijklmnopqrst11', #2, 'Waschbecken_1 70', $,
'IfcFlowTerminal', (#10), (#1011), $, $, .URINAL.);
#10=IFCPROPERTYSET('abcdefghijklmnopqrst12', #2, 'FM Parameter', $, (#11,#12,#13));
#11=IFCPROPERTYSINGLEVALUE('Group', $, IFCLABEL('Sanit\S\dre Anlagen'), $);
#12=IFCPROPERTYSINGLEVALUE('Production year', $, IFCNUMERICMEASURE(1998), $);
#13=IFCPROPERTYSINGLEVALUE('Brutto Gewicht', $, IFCNUMERICMEASURE(1.52), $);

/* definition of the occurrences  -no geometry shown- */
#1000=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst17', #2, $, $, (#14), #1);
#14=IFCFLOWTERMINAL('abcdefghijklmnopqrst13', #2, $, $, $, #101, #102, $);

/* occurrence properties */
#2028=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst23', #2, $, $, (#14), #2032);
#2032=IFCPROPERTYSET('abcdefghijklmnopqrst24', #2, 'FM occurrence parameter', $,
(#2034,#2033));
#2033=IFCPROPERTYSINGLEVALUE('Asset number', $, IFCLABEL('DE_A_234234'), $);
#2034=IFCPROPERTYSINGLEVALUE('Serial number', $, IFCLABEL('BD124-GF5835'), $);

/* overridden shared properties */
#2128=IFCRELOVERRIDESPROPERTIES('abcdefghijklmnopqrst45', #2, $, $, (#14), #10,
(#2130));
#2130=IFCPROPERTYSINGLEVALUE('Production year', $, IFCNUMERICMEASURE(1999), $);
```

6.1.2.3 Concepts of the performance history control

Building service occurrences come into existence at different phases of the building-lifecycle. Furthermore, they have properties that change over time, such as the temperature of the air in a duct. The performance history concept was introduced in IFC2x2 to capture such properties independently from the occurrence element itself. This allows for the possibility of the exchange file to capture all known historical information associated with an element. As a subtype of *IfcControl*, *IfcPerformanceHistory* entities can be related to subtypes of *IfcFlowDistributionElement* via the *IfcRelAssignsToControl* relationship. The *IfcPerformanceHistory* entity handles the:

- type name as *IfcObject.Name*
- description of the type as *IfcObject.Description*
- lifecycle phase using the *IfcPerformanceHistory.LifeCyclePhase* attribute
- property sets using *IfcObject.IsDefinedBy* and the *IfcRelDefines* relationship

6.2 Concept of connectivity

Connectivity between building service and electrical components is an important concept within IFC2x2. There are two ways to handle connectivity:

- as a logical connectivity concentrating on the fact that two components connect via their ports
- as a physical connectivity where the physical manifestation of the connection is explicitly described as well

6.2.1 Concept of Logical Connectivity

The concept of connectivity is generally associated with things that are directly or physically connected. However, there are circumstances where it is important to know that there is a connective relationship between two (or more) objects but where there is no physical connection to identify this fact. A simple idea of this could be seen in the *IfcRelContainedInSpatialStructures* relationship. Whilst this is not intended to show connectivity, it does identify the connection of an element within a particular spatial structure.

The idea of non-physical connection may be termed logical connectivity in the sense that there is a logical or implied connection between objects. Logical connectivity may be achieved using instances of *IfcPort* (or its subtype *IfcDistributionPort*) or it may need to be handled by more general connectivity concepts where ports are not present or cannot be used. The use of ports for connectivity is the normal way to handle connectivity for building service components.

6.2.1.1 Logical Connectivity With Ports

A particular example of logical connectivity that can be achieved using ports is the connection between a light switch and the lights that are turned on and off by that switch. Frequently, the cable that connects the switch with a light will not be instantiated in a model and so the physical connection cannot be achieved. In this case, the cable making the connection has to be implied.

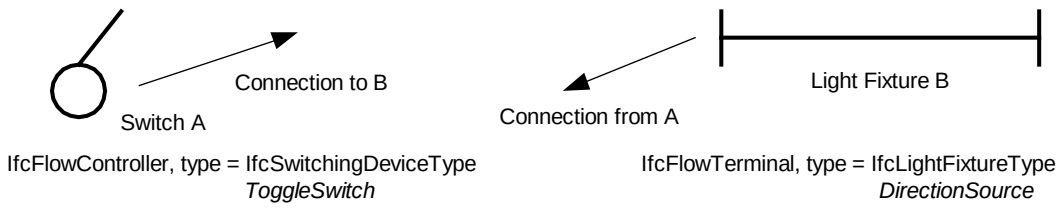


Figure 67: Logical connection between switch and light fixture

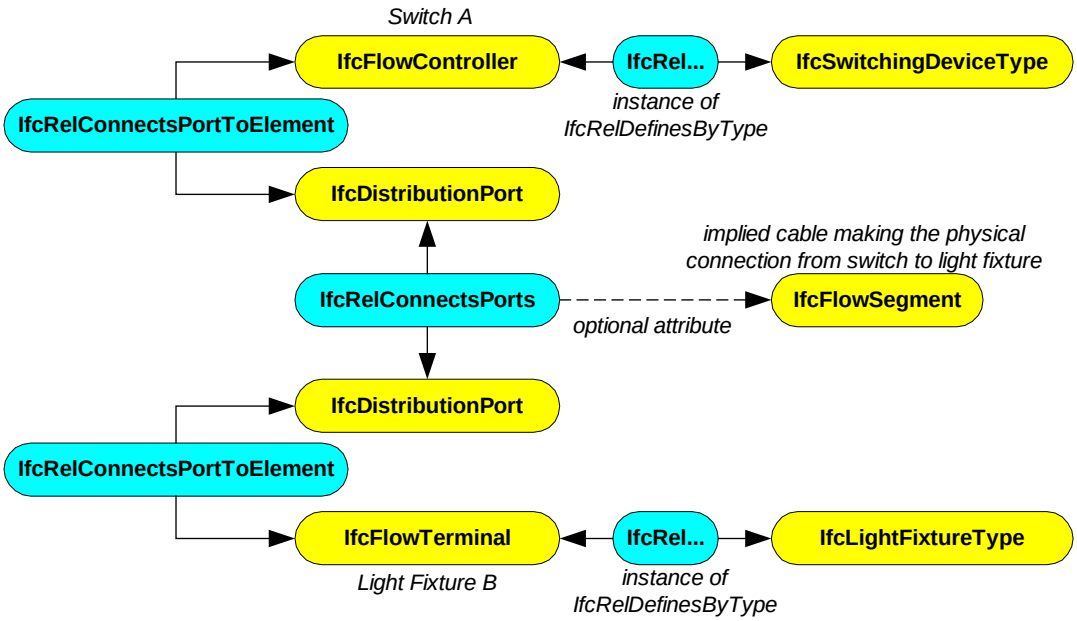


Figure 68: Entities used in logical connectivity with ports

Because both the light switch (as an instance of *IfcFlowController*) and the light fixture (as an instance of *IfcFlowTerminal*) may have related instances of *IfcDistributionPort* (bound through the *IfcRelConnectsPortToElement*), then logical connection may be achieved using ports.

IfcRelConnectsPorts has an additional attribute, *RealizingElement*, that enables the specification of a realizing element. In this case, the realizing element is the implied cable connection between the switch and the light fixture. This can be instantiated although, since the connection is logical, it cannot have a representation. Similarly, the implied connection is made as a single occurrence; it is not possible with such a logical connection to express in detail a cable routing. Strictly speaking: the instantiation of the realizing element would create a physical connectivity (see 6.2.2).

6.2.1.2 Logical Connectivity Without Ports

An example of a logical connectivity without ports may be seen from the situation of a water supply pipe taken to discharge into a tank (the particular situation in this example being a potable water pipe which is taken to discharge into a tank containing non potable water in which case, a specific distance between the outlet of the potable discharge and the waterline of the non potable water in the tank must be maintained).

In this case, the discharge end of the pipe will have a distribution port that is unfulfilled (i.e. it does not otherwise participate in a connection relationship). However, the surface of the tank water content does not contain a port to which a connection can be made.

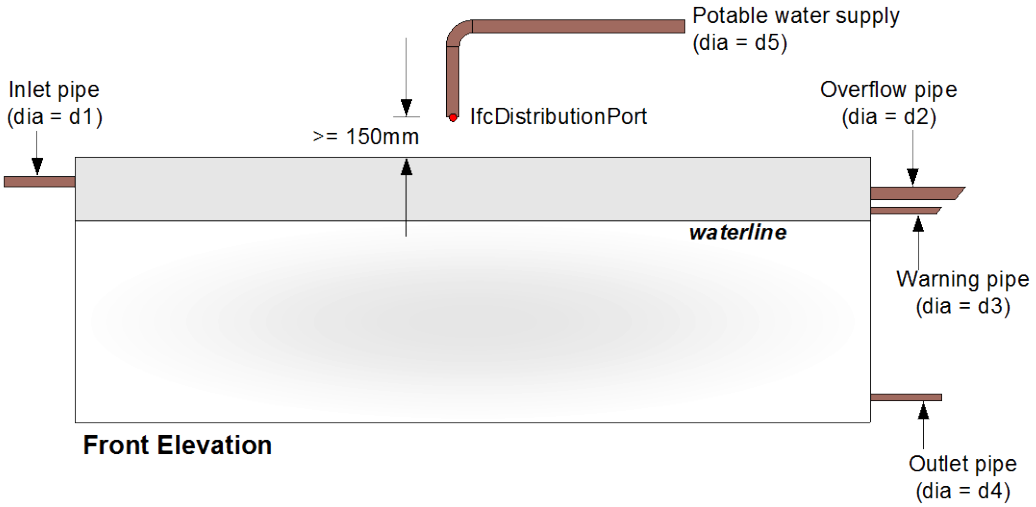


Figure 69: Tank with logical connection between discharge and waterline using an air gap

Theoretically, a port could be introduced at the waterline of the tank and a connection made using *IfcRelConnectsPorts*. It is considered unlikely that this facility will be offered in software and, in fact, is considered to be undesirable since it would break the useful concept that an unfulfilled port describes an open end in a pipe.

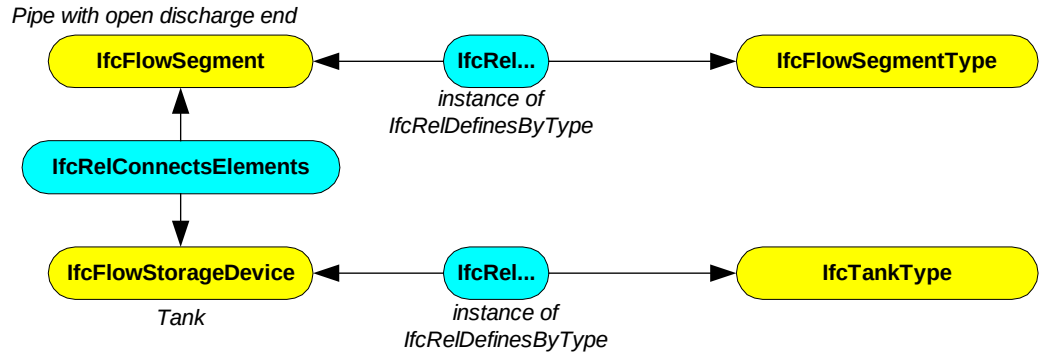


Figure 70: Defining logical connectivity without ports

Sine the waterline does not have a port, an alternative connection needs to be defined. This can be achieved using the *IfcRelConnectsElements* relationship. In this case, the elements concerned will be the flow segment describing the open ended pipe and the tank. Questions of the distance from the discharge end of the pipe to the water surface have to be answered by deriving appropriate coordinate values from the positioning data available.

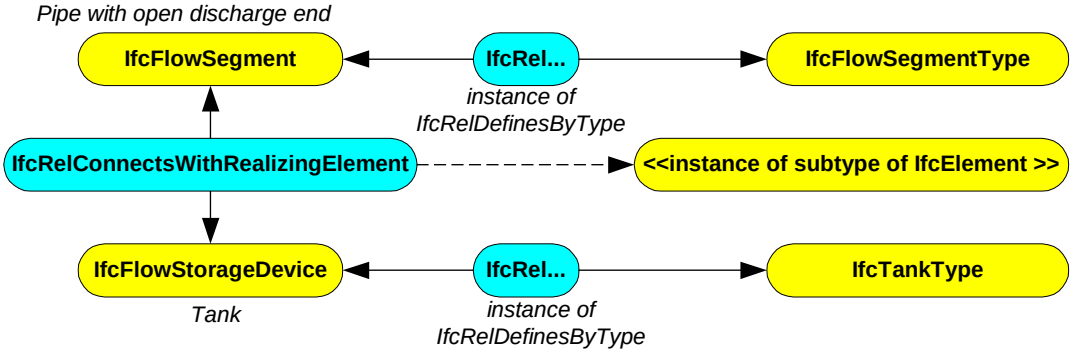


Figure 71: Alternative logical connectivity without ports using a realizing element

An alternative approach would be to use *ifcRelConnectsWithRealizingElement* in which case an appropriate entity for the object realizing the connection relationship would need to be determined.

6.2.2 Concept of physical connectivity

If the logical connectivity (as described in section 6.2.1) is enhanced by a realizing element, it constitutes a physical connectivity.

The new IFC2x2 substantially enhances the ability to traverse connected building services elements. This is handled primarily through the *IfcDistributionPort* and *IfcRelConnectsPorts* and *IfcRelConnectsPortToElement* relationships. The following figures exemplify these using a duct segment connecting to a tee and distributing air out through two connecting duct segments as shown in Figure 71.

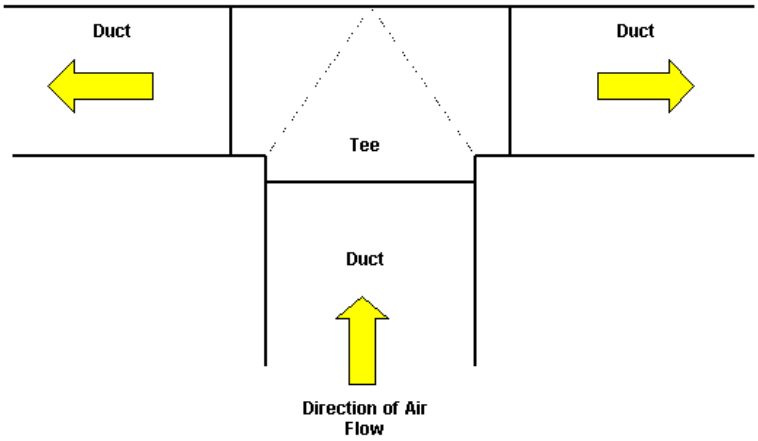


Figure 72 : Duct segments connected to a tee fitting

Any building services element can have a related *IfcDistributionPort* instance related to it via the *IfcRelConnectsPortToElement* relationship (see Figure 72). Although the port has a local placement, which allows it to be aligned with its related element, it typically does not have any physical manifestation, and consequently there is no corresponding type definition for the port concept.

Use the *IfcDistributionPort.Name* attribute to define the sequential port number assigned to each element. This is necessary for relating property set values that contain lists of properties associated with a port, such as connection type, size, etc. Ports are connected together using the *IfcRelConnectsPorts* relationship. The relationship can also utilize the *IfcRelConnectsPorts.RealizingElement* attribute to relate an element instance, such as an *IfcFlowFitting*, which is acting as a coupling or other type of connector entity.

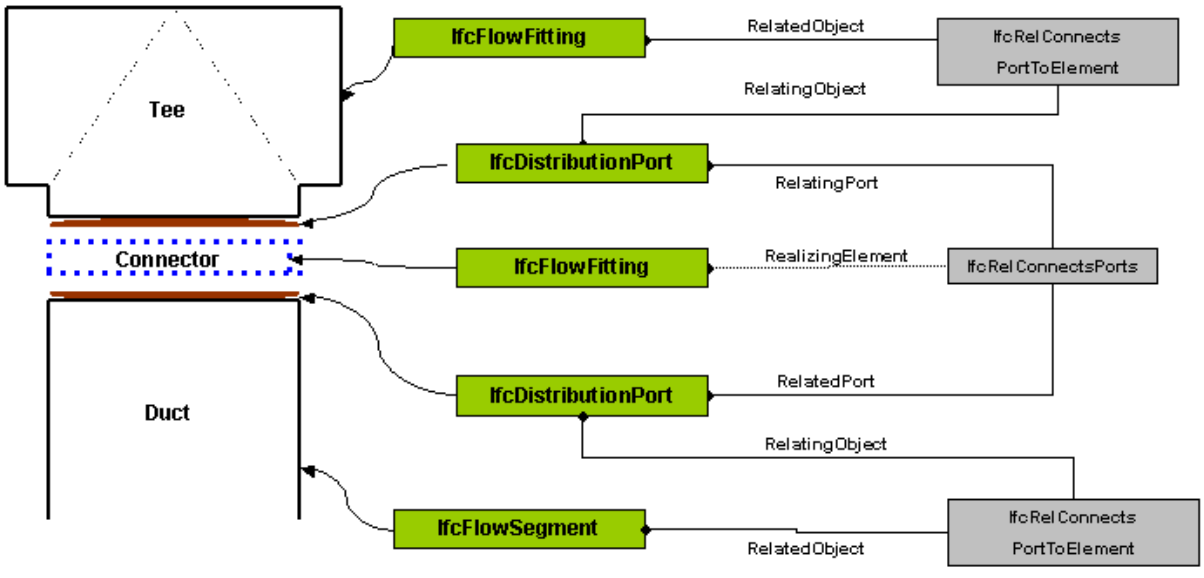


Figure 73 : Connectivity example between two building services elements

Relating other engineering properties, such as flow characteristics, etc. to the port is accomplished as shown in Figure 73. Note that instances of *IfcFluidFlowProperties* and property set information containing connection-specific parameters are shared between both ports rather than duplicated.

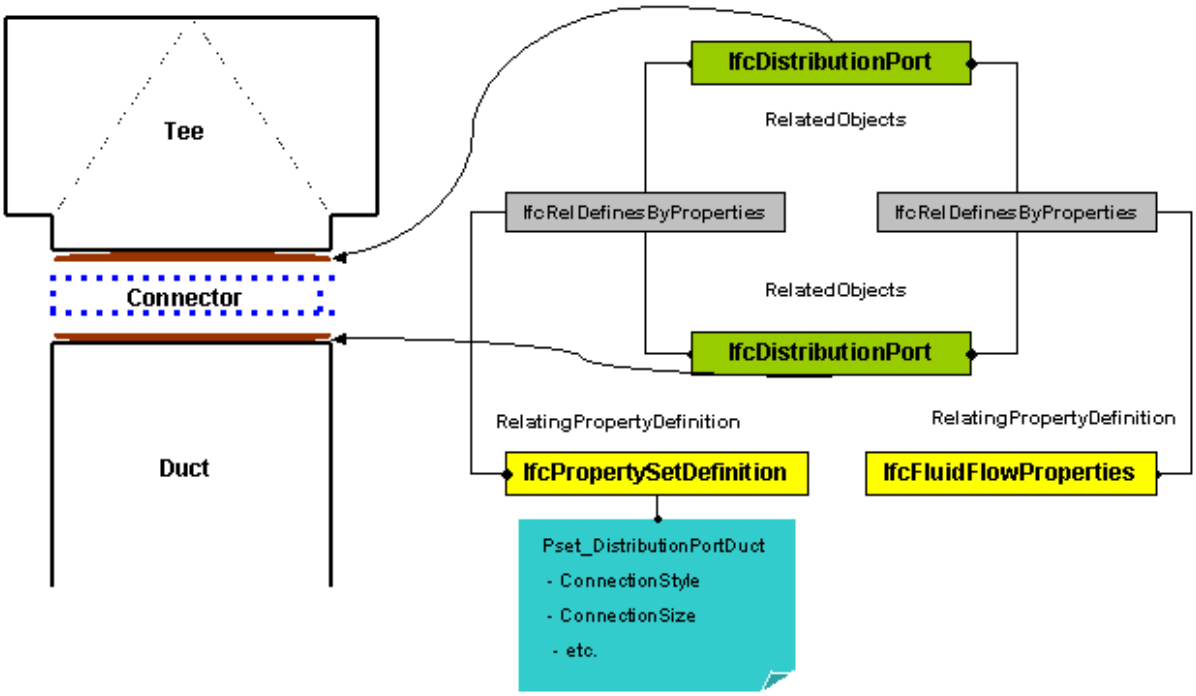


Figure 74 : Assigning flow characteristics to ports

6.3 Specific concepts for selected building service elements

There are specific guidelines available to handle different building service elements. Currently included within this section of the Model Implementation Guide are:

- flow moving devices
- energy conversion devices
- flow storage devices

6.3.1 Concept of Flow Moving Device

An *IfcFlowMovingDevice* is defined as a device that participates in a distribution system and is used for moving or impelling a fluid, such as a liquid or a gas, around a system. Generally, a flow moving device may be thought of as a fan, pump, or compressor.

As is normal for components in building services systems, flow moving devices are specified by type through a type entity (e.g. *IfcPumpType*) with each occurrence of a flow moving device specified using *IfcFlowMovingDevice*. The occurrence of *IfcFlowMovingDevice* is typed as e.g. a pump through an instance of the *IfcRelDefinesByType* relationship class.

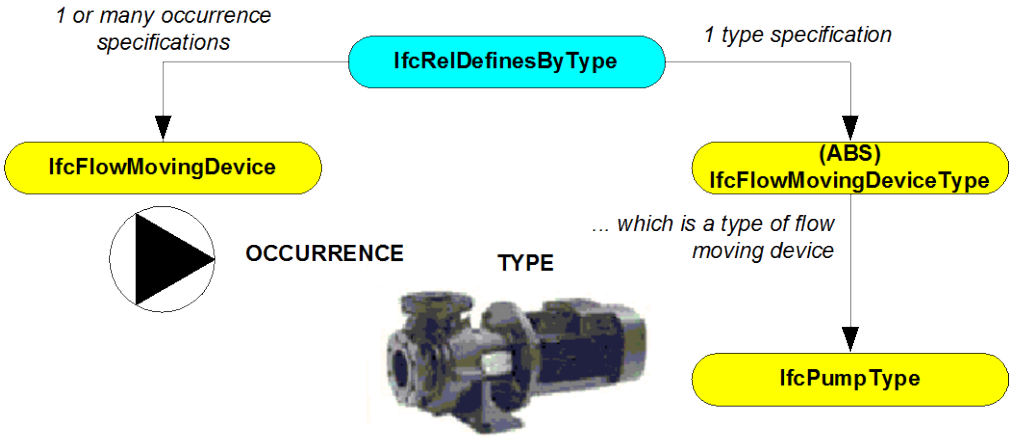


Figure 75: Pump type and occurrence definition

```
/* definition of the type -no geometry shown or property sets defined- */
#1= IFCPUMPTYPE('abcdefghijklmnopqrst11', #2, 'Pump_1', $, 'IfcFlowMovingDevice', $,
(#1011), $, $, .CIRCULATOR.);

/* definition of the occurrences  -no geometry shown- */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $, (#20), #1);

#20=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);
```

6.3.1.1 Creating Pump Aggregations

Pumps are typically installed complete with a set of isolating valves that enable the pump to be removed for maintenance or exchange. It is possible to make an aggregation of the physical pump and the valves (and any connecting pipework and fittings) and treat them as though they were a single object. In this case, the object that is the target of the aggregation becomes also an *IfcFlowMovingDevice*. The bringing together of the objects into an aggregation is achieved using the relationship class *IfcRelAggregates*.

```
/* definition of the occurrences being aggregated -no geometry shown- */
#20=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);
#21=IFCFLOWCONTROLLER('abcdefghijklmnopqrst21', #2, $, $, $, #10003, #10004, $);
#22=IFCFLOWCONTROLLER('abcdefghijklmnopqrst22', #2, $, $, $, #10005, #10006, $);
#23=IFCFLOWFITTING('abcdefghijklmnopqrst23', #2, $, $, $, #10007, #10008, $);
#24=IFCFLOWFITTING('abcdefghijklmnopqrst24', #2, $, $, $, #10009, #10010, $);
#25=IFCFLOWFITTING('abcdefghijklmnopqrst25', #2, $, $, $, #10011, #10012, $);
#26=IFCFLOWFITTING('abcdefghijklmnopqrst26', #2, $, $, $, #10013, #10014, $);

/* defines the pump aggregation */
#50=IFCRELAGGREGATES('abcdefghijklmnopqrst50', #2, $, $, #60,
(#20,#21,#22,#23.#24,#25,#26));

/* specification of the aggregation – without placement or representation*/
#60=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst60', #2, $, $, $, $, $, $);

/* definition of occurrence property set */
#300=IFCPROPERTYSET('abcdefghijklmnopqrst300', #2, 'Pset_FlowMovingDevicePump', $,
(#310,#311,#312));
#301=IFCPROPERTYSET('abcdefghijklmnopqrst301', #2, 'Pset_FlowMovingDevicePump', $,
(#320,#321,#322));

/* defines specific properties for a pump */
#310=IFCPROPERTYENUMERATEDVALUE('PumpComposition', $, ('ELEMENT'), #351);
#311= …… ;
#312= …… ;
#320=IFCPROPERTYENUMERATEDVALUE('PumpComposition', $, ('COMPLEX'), #351);
#321= …… ;
#322= …… ;

/* defines property enumerations used in specific type property set */
#351=IFCPROPERTYENUMERATION('PEnum_PumpComposition', ('COMPLEX', 'ELEMENT', 'PARTIAL',
'NOTKNOWN', 'UNSET'), $);
```

```
/* definition of properties for the occurrences */
#410=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst410', #2, $, $, (#20), #300);
#411=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst411', #2, $, $, (#60), #301);
```

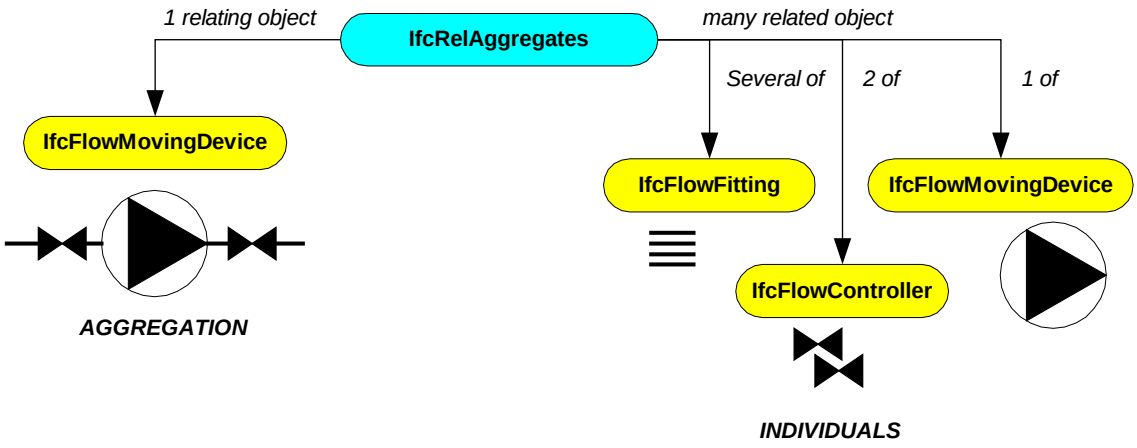


Figure 76: Specifying a pump aggregation

As with flow storage devices (tanks), pumps can be identified as being elemental or complex through the property *PumpComposition* contained in the property set *Pset_FlowMovingDevicePump*.

Note that this property set is defined for pump occurrences and not for pump types.

6.3.1.2 Creating Pump Sets

A pump set is considered to be an aggregation of 2 or more pumps that can act either together in series or in parallel or on a run and standby basis (i.e. one pump switched on and working whilst a second pump, and typically same sized pump, is inoperative but ready to become operative in the case of the first pump failing or being taken out of action.

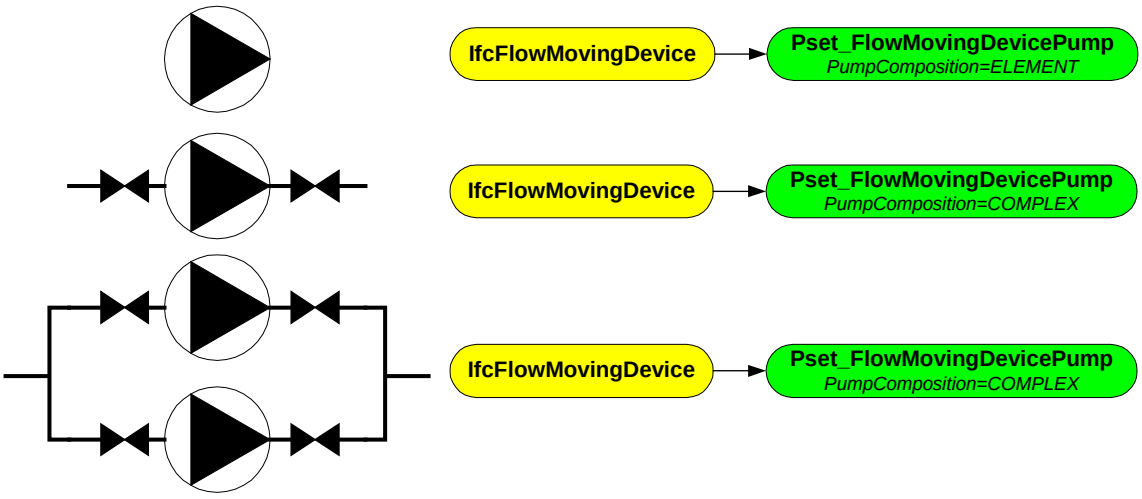


Figure 77: Different levels of pump composition

In this case, the pump set aggregation consists of two or more of the pump aggregations as above (comprising the pump, valves, pipework and fittings), the valve, pipework and fitting that may be installed as a bypass line (i.e. a line forming part of the pump set but in which flow can bypass the installed pumps) and any pipework and fittings used to bring all of the individual aggregations together into a total aggregation.

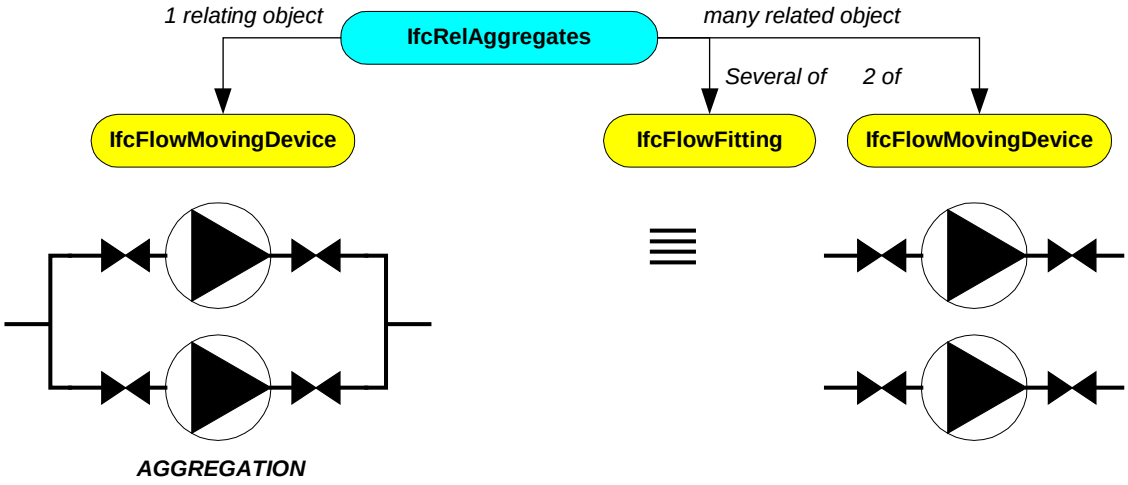


Figure 78: Specifying a pump set aggregation

```
/* definition of the occurrences being aggregated -no geometry shown- */
#41=IFCFLOWFITTING('abcdefghijklmnopqrst23', #2, 'Bend', $, $, #10061, #10062, $);
#42=IFCFLOWFITTING('abcdefghijklmnopqrst24', #2, 'Bend', $, $, #10063, #10064, $);
#43=IFCFLOWFITTING('abcdefghijklmnopqrst23', #2, 'Bend', $, $, #10065, #10066, $);
#44=IFCFLOWFITTING('abcdefghijklmnopqrst24', #2, 'Bend', $, $, #10067, #10068, $);
#45=IFCFLOWFITTING('abcdefghijklmnopqrst25', #2, 'Tee', $, $, #10069, #10070, $);
#46=IFCFLOWFITTING('abcdefghijklmnopqrst26', #2, 'Tee', $, $, #10071, #10072, $);

/* definition of the pump set aggregation */
#51IFCRELAGGREGATES('abcdefghijklmnopqrst51, #2, $, $, #70,
(#41,#42,#43.#44,#45,#46,#60,#61));

/* specification of the pump aggregations */
#60=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst60', #2, 'Pump Aggregation',$,$,$,$,$);
#61=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst61', #2, 'Pump Aggregation',$,$,$,$,$);

/* specification of the pump set aggregation */
#70=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst70', #2, 'Pump Set Aggregation',$,$,$,$,$);

/* definition of occurrence property sets */
#301=IFCPROPERTYSET('abcdefghijklmnopqrst301', #2, 'Pset_FlowMovingDevicePump', $,
(#320,#321,#322));
#302=IFCPROPERTYSET('abcdefghijklmnopqrst302', #2, 'Pset_FlowMovingDevicePump', $,
(#320,#321,#322));
#303=IFCPROPERTYSET('abcdefghijklmnopqrst303', #2, 'Pset_FlowMovingDevicePump', $,
(#320,#321,#322));

/* definition of specific properties for a pump */
#320=IFCPROPERTYENUMERATEDVALUE('PumpComposition', $, ('COMPLEX'), #351);
#330=IFCPROPERTYENUMERATEDVALUE('PumpComposition', $, ('COMPLEX'), #351);
#340=IFCPROPERTYENUMERATEDVALUE('PumpComposition', $, ('COMPLEX'), #351);

/* definition of property enumerations used in specific type property set */
#351=IFCPROPERTYENUMERATION('PEnum_PumpComposition', ('COMPLEX', 'ELEMENT', 'PARTIAL',
'NOTKNOWN', 'UNSET'), $);

/* defines property relationships for the pump aggregation occurrences */
#411=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst411', #2, $, $, (#60), #301);
#412=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst412', #2, $, $, (#61), #302);

/* defines property relationship for the pump set aggregation occurrence */
#413=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst413', #2, $, $, (#70), #303);
```

6.3.1.3 Common Properties of Pump Types

The common properties of a pump type are given in *Pset_PumpTypeCommon* and includes such ideas as: *FlowRateRange, FlowResistanceRange, ConnectionSize, CasingMaterial, ImpellerMaterial, ImpellerSealMaterial, TemperatureRange, NetPositiveSuctionHead, NominalRotationSpeed.*

Note that the values for flow rate; flow resistance and temperature range are specified as bounded values (which means that an upper and/or lower limit value can be set).

6.3.1.4 Specifying The Pump Motor

A pump is typically driven by an electric motor. There are two ways in which a pump motor can be specified:

1. directly at the pump by using a property set for the motor and defining it directly for the pump (most frequently used for a close coupled pump),
2. indirectly through a motor connection.

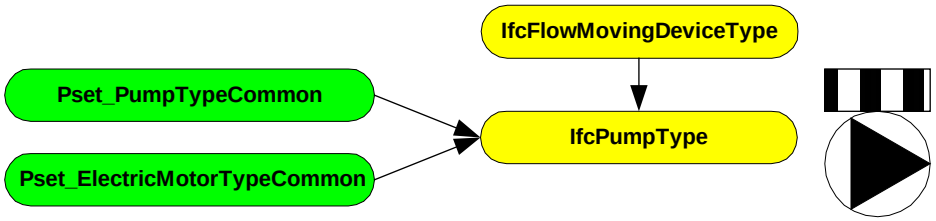


Figure 79: Directly specifying a pump motor

Where the motor is specified indirectly, type and occurrence specifications are required for the pump, the motor and the connection made between the pump and the motor.

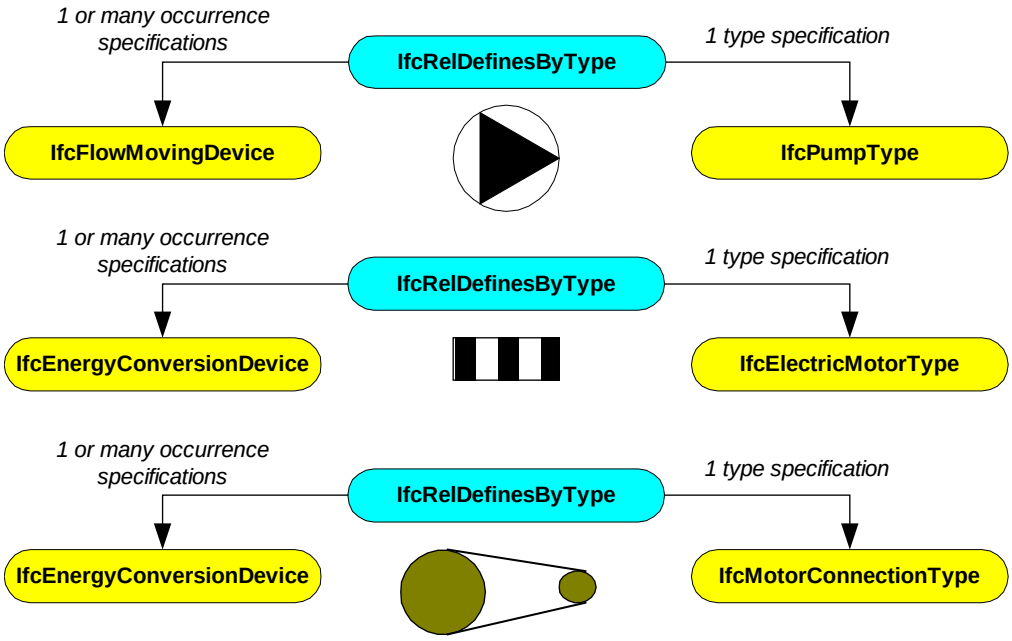


Figure 80: Types and occurrences for indirect motor connection

When the occurrences have been specified, they can be connected together using the relationship *IfcRelConnectsWithRealizingElement*. This is a clear example of where the ternary or third relationship capability can be used since the specific intention of a motor connection is to realize the connection made between a motor as the driving device and (in this case) the pump as the driven device. By so doing, only one relationship is required. If the connections were made using other, conventional connection relationships, two separate relationships would be needed, one for the connection between motor and motor connection and one between the pump and the motor connection.

The actual type of the motor connection can be specified by the enumeration value *IfcMotorConnectionTypeEnum* that allows identification of whether direct drive, belt drive or coupling (e.g. fluid or viscous coupling) is used.

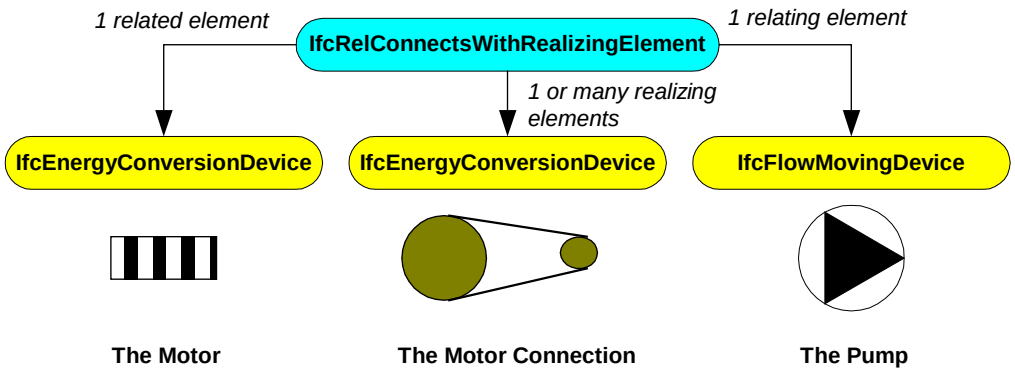


Figure 81: Realizing the indirect motor connection

```
/* definition of the types -no geometry shown or property sets defined- */
#1= IFCPUMPTYPE('abcdefghijklmnopqrst01', #2, 'Pump_1', $, 'IfcFlowMovingDevice', $,
(#1011), $, $, .ENDSUCTION.);
#2= IFCELECTRICMOTORTYPE('abcdefghijklmnopqrst02', #2, 'Motor_1', $,
'IfcEnergyConversionDevice', $, (#1012), $, $, .INDUCTION.);
#3= IFCMOTORCONNECTIONTYPE('abcdefghijklmnopqrst03', #2, 'Connection_1', $,
'IfcEnergyConversionDevice', $, (#1013), $, $, .BELTDRIVE.);

/* defines the type relationships */
#11=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst11', #2, $, $, (#21), #1);
#12=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst12', #2, $, $, (#22), #2);
#13=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst13', #2, $, $, (#23), #3);

/* defines the type occurrences */
#21=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst21', #2, $, $, $, #10001, #10002, $);
#22=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst22', #2, $, $, $, #10003, #10004, $);
#23=IFCENERGYCONVERSIONDEVICE ('abcdefghijklmnopqrst23', #2, $, $, $, #10005, #10006,
$);

/* realizes the connection relationship */
#30=IFCRELCONNECTSWITHREALIZINGELEMENT ('abcdefghijklmnopqrst30', #2, $, $, $, #21, #22,
(#23), $);
```

6.3.2 Concept of Energy Conversion Device

An *IfcEnergyConversionDevice* is defined as a device that is used to perform energy conversion or heat transfer and that typically participates in a flow distribution system. In the context of general energy conversion, it may be used to identify a device that converts energy from one form to another such as an electric motor. In the context of heat transfer, it may be used to identify a device in which heat is transferred from a primary medium to a secondary medium such as a boiler, water heater, chiller etc.

In the heat transfer context, energy conversion is concerned both with the circumstance in which the secondary medium is heated (boiler, water heater) and that in which the secondary medium is cooled (chiller).

As is normal for components in building services systems, an energy conversion device is specified by type through *IfcEnergyConversionDeviceType* with each occurrence of e.g. a boiler specified using *IfcEnergyConversionDevice*.

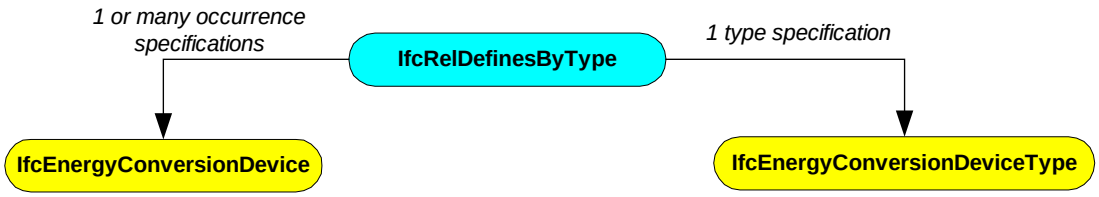


Figure 82: Energy conversion device type and instance

See Table 8 for particular types of energy conversion device that are defined as subtypes of *IfcEnergyConversionDeviceType*. The Figure 82 shows the subtypes:

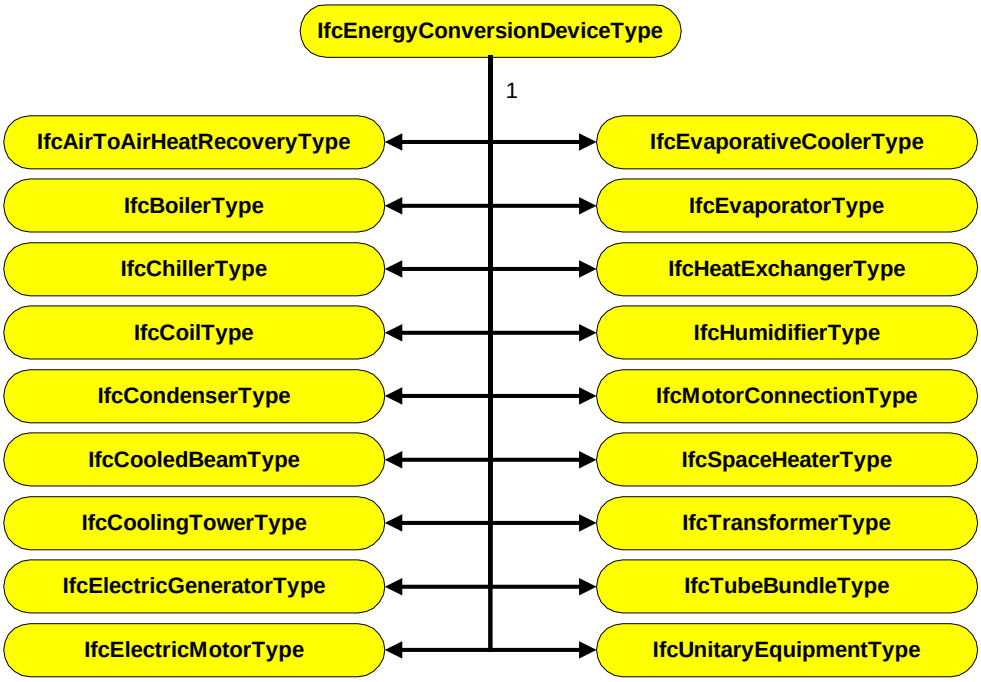


Figure 83: Hierarchy of IfcEnergyConversionDeviceType

```
/* definition of the type -no geometry shown or property sets defined- */
#1= IFCBOILERTYPE('abcdefghijklmnopqrst01', #2, 'Boiler_1', $,
'IfcEnergyConversionDevice', $, (#1011), $, $, .WATER.);

/* definition of the occurrences  -no geometry shown- */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $, (#20), #1);

#20=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);
```

Energy conversion devices handle a substantial number of concepts in building services. Some particular issues that can arise and how they may be dealt with in an IFC file are discussed further below.

6.3.2.1 Identifying Water Heaters

In general, a water heater may be defined as an energy conversion device. In the sense that it heats water, it may be considered to be a boiler whose predefined type is ‘water’.

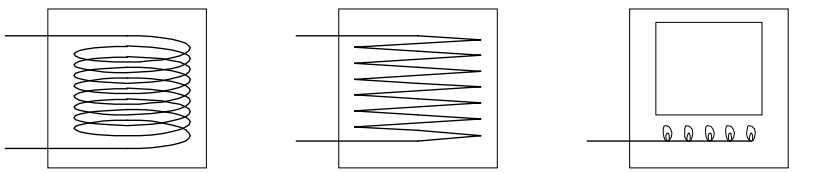


Figure 84: Types of water heater

Various different forms of water heater may be defined based on the mechanism used for heating the water (the primary energy source) and whether the water is to be heated and stored (storage type) or made available for instant use (instantaneous type).

6.3.2.1.1 Property Sets Associated With a Water Heater

More detail about a specific type of water can be given in the *Pset_BoilerTypeCommon* property set that is attached at the type entity.

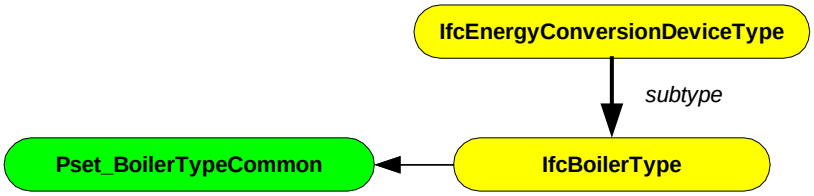


Figure 85: Applying the common property set to a boiler type

Property	Property Type	Value
PrimaryMedium	Enumerated Value	GAS
PressureRating	Single Value	-
OperatingMode	Enumerated Value	UNSET
Material	Reference	Steel
PrimaryEnergySource	Reference	-
AuxiliaryEnergySource	Reference	-
HeatTransferSurfaceArea	Single Value	-
NominalPartLoadRatio	Bounded Value	-
WaterInletTemperatureRange	Bounded Value	15C/18C
WaterStorageCapacity	Single Value	60 litres
Weight	Single Value	100 kg
SoundLevel	Reference	-
PartialLoadEfficencyCurves	List Value	-
NominalEfficiency	List Value	-
HeatOutput	List Value	-
OutletTemperatureRange	Bounded Value	35C/40C
NominalEnergyConsumption	Single Value	-

Table 1: Values in Pset_BoilerTypeCommon

In this case, because gas is used as the primary medium, the water heater is considered to be 'Direct' fired. This would be the case also if the primary medium were 'Electric'. For indirect heaters using heat transfer from water or steam, the values would be prefixed by 'Indirect' (as in INDIRECT_WATER, INDIRECT_STEAM).

```
/* definition of the type -no geometry shown - */
#1= IFCBOILERTYPE('abcdefghijklmnopqrst01', #2, 'Boiler_1', $,
'IfcEnergyConversionDevice', (#5001), (#1011), $, $, .WATER.);

/* definition of the occurrences  -no geometry shown- */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $, (#20), #1);

#20=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);

/* definition of the common property set and reference to the individual properties - */
#5000=IFCPROPERTYSET('abcdefghijklmnopqrst5000', #2, 'Pset_BoilerTypeCommon', 'Common
attributes for a water heater', (#5001, #5002, #5003, #5004, #5005, #5006, #5007, #5008,
#5009, #5010, #5011, #5012, #5013, #5014, #5015, #5016, #5017));
```

```
/* the properties … */
#5001=IFCPROPERTYENUMERATEDVALUE('PrimaryMedium', $, ('GAS'), $);
#5002=IFCPROPERTYSINGLEVALUE('PressureRating', $, 0.0, $);
#5003=IFCPROPERTYENUMERATEDVALUE('OperatingMode', $, ('UNSET'), $);
```

In the above example, the property set is partially expanded to show some of the properties. In this example, the actual enumeration value used for the primary medium and the operating mode is shown but the enumeration references from which they are taken are not quoted.

6.3.2.1.2 Water Heaters as Electrical Appliances

It should be noted that the IFC model also contains the concept of a water heater as an electrical appliance (as a value within the *IfcElectricalApplianceTypeEnum*). This should be used only for examples of small, movable water heaters that have local plug connections (i.e. are not directly wired to an electrical system or that do not use plugs that are expected to be permanently placed). Such a water heater will also not use the *Pset_BoilerTypeCommon* for attribute information.

Other than as above, the *IfcElectricalApplianceTypeEnum=.WATERHEATER.* should not be used.

6.3.2.2 Air Handling Units

Air handling units (air handlers) are instances of *IfcUnitaryEquipmentType* used for the purpose of changing air conditions (humidity and/or temperature) between the entry and exit points.

Air handling units may be supplied by a manufacturer as a complete package or constructed of discrete components. Even when an air handling unit is delivered as a complete package, there may be a need to deal with information at a component based level for design or maintenance purposes. Therefore, it is necessary to be able to handle information directly at the component level and at the aggregated air handling unit level. The aggregation is achieved using *IfcRelAggregates*.

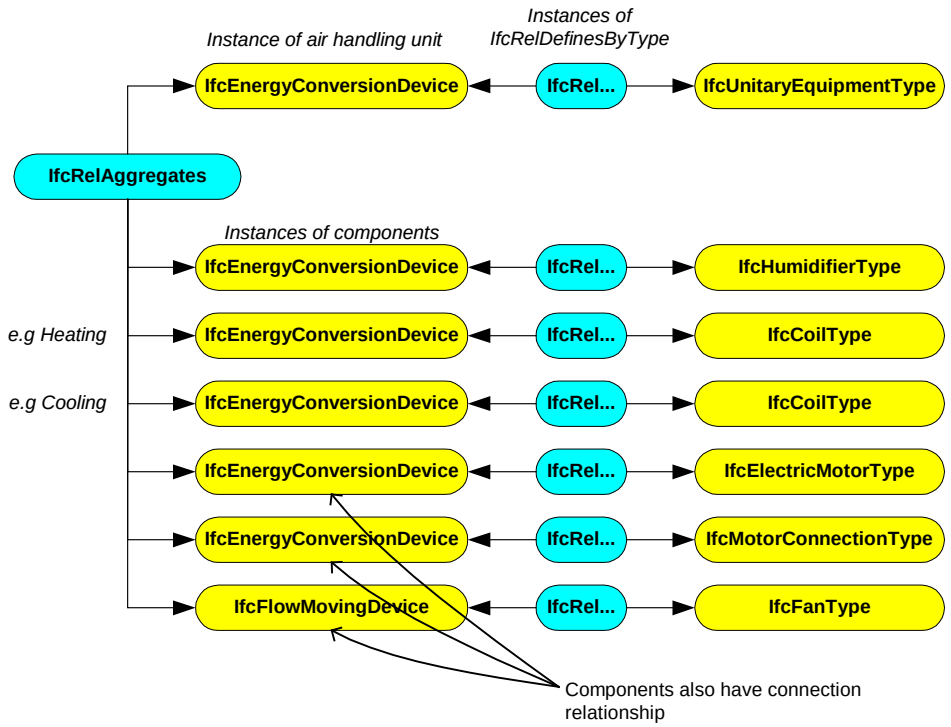


Figure 86: Aggregation of components in an air handling unit

6.3.2.2.1 Specifying The Fan Motor

The fan, the motor and the connection between them can be connected together using the relationship *IfcRelConnectsWithRealizingElement*. The type of the motor connection can be specified by the enumeration value *IfcMotorConnectionTypeEnum* that allows identification of whether direct drive, belt drive or coupling (e.g. fluid or viscous coupling) is used. See also 6.3.1.4.

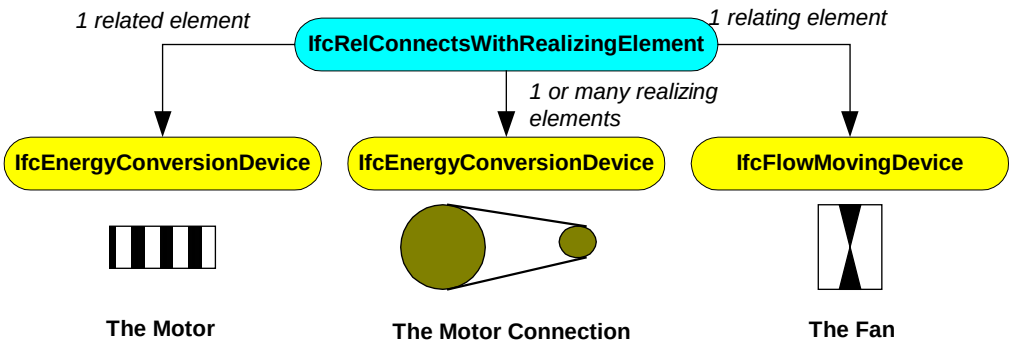


Figure 87: Realizing the indirect motor connection

```
/* definition of the types -no geometry shown or property sets defined- */
#1= IFCFANTYPE('abcdefghijklmnopqrst01', #2, 'Fan_1', $, 'IfcFlowMovingDevice', $,
(#1011), $, $, .CENTRIFUGALFORWARDCURVED.);
#2= IFCELECTRICMOTORTYPE('abcdefghijklmnopqrst02', #2, 'Motor_1', $,
'IfcEnergyConversionDevice', $, (#1012), $, $, .INDUCTION.);
#3= IFCMOTORCONNECTIONTYPE('abcdefghijklmnopqrst03', #2, 'Connection_1', $,
'IfcEnergyConversionDevice', $, (#1013), $, $, .BELTDRIVE.);

/* defines the type relationships */
#11=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst11', #2, $, $, (#21), #1);
#12=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst12', #2, $, $, (#22), #2);
#13=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst13', #2, $, $, (#23), #3);

/* defines the type occurrences */
#21=IFCFLOWMOVINGDEVICE('abcdefghijklmnopqrst21', #2, $, $, $, #10001, #10002, $);
#22=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst22', #2, $, $, $, #10003, #10004, $);
#23=IFCENERGYCONVERSIONDEVICE ('abcdefghijklmnopqrst23', #2, $, $, $, #10005, #10006,
$);

/* realizes the connection relationship */
#30=IFCRELCONNECTSWITHREALIZINGELEMENT ('abcdefghijklmnopqrst30', #2, $, $, $, #21, #22,
(#23), $);
```

6.3.3 Concept of Flow Storage Device

An *IfcFlowStorageDevice* is defined as a device that participates in a distribution system and is used for temporary storage of a fluid such as a liquid or a gas. Generally, a flow storage device may be thought of as a tank although other types of storage device could be specified.

6.3.3.1 Tanks as Flow Storage Devices

A tank is the most general form of flow storage device. As is normal for components in building services systems, tanks are specified by type through *IfcTankType* with each occurrence of a tank specified using *IfcFlowStorageDevice*. The occurrence of *IfcFlowStorageDevice* is typed as a tank through an instance of the *IfcRelDefinesByType* relationship class.

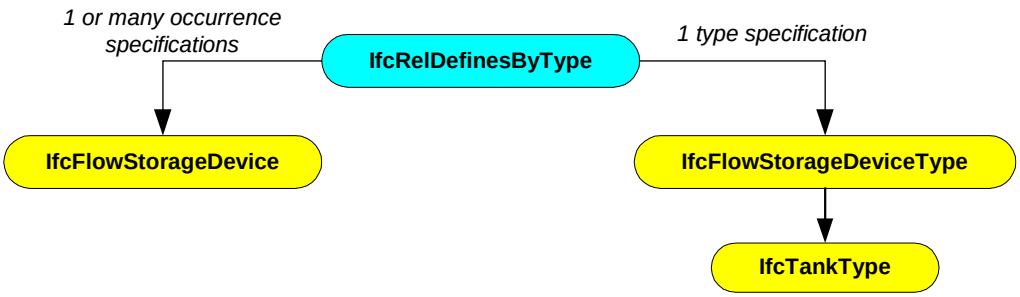


Figure 88: Flow storage device type and occurrence

```
/* definition of the type -no geometry shown or property sets defined- */
#1= IFCTANKTYPE('abcdefghijklmnopqrst11', #2, 'Tank_1', $, 'IfcFlowStorageDevice', $,
(#1011), $, $, .PREFORMED.);

/* definition of the occurrences  -no geometry shown- */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $, (#20), #1);
#20=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);
```

6.3.3.2 Property Sets Associated With a Tank Type

More detail about a specific type of tank can be given in property sets that are attached at the type entity.

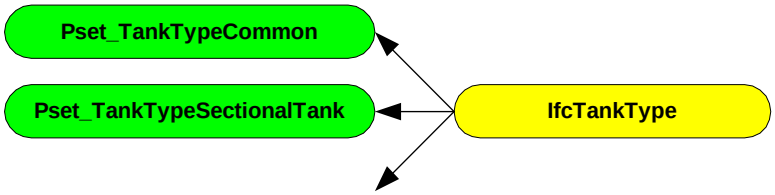


Figure 89: Property sets associated with a tank type

Several possibilities are defined as standard property sets within the IFC model:

- Properties Common To All Types of Tank
  - *Pset_TankTypeCommon*
- Properties Specific to A Type of Tank
  - *Pset_TankTypeExpansion*
  - *Pset_TankTypePressureVessel*
  - *Pset_TankTypePreformedTank*
  - *Pset_TankTypeSectionalTank*

As an example, consider a storage tank with a nominal capacity of 6000 litres. The common properties given within the *Pset_TankTypeCommon* for such a tank are:

Property	Property Type	Value
Type	Enumerated Value	WATERSTORAGEPOTABLE
AccessType	Enumerated Value	LOOSECOVER
NominalLengthOrDiameter	Single Value	3m
NominalWidthOrDiameter	Single Value	2m
NominalDepth	Single Value	1m
NominalCapacity	Single Value	6 m^3 (6000 litres)
EffectiveCapacity	Single Value	4.6 m^3 (4600 litres)
OperatingWeight	Single Value	4950 kg
Material	Reference	Steel
MaterialThickness	Single Value	2mm

Table 2: Properties for Tanks as defined in Pset_TankTypeCommon

6.3.3.2.1 Tank Shape

Geometrically, a tank may be specified as rectangular, as a horizontal or vertical cylinder or as a spherical container.

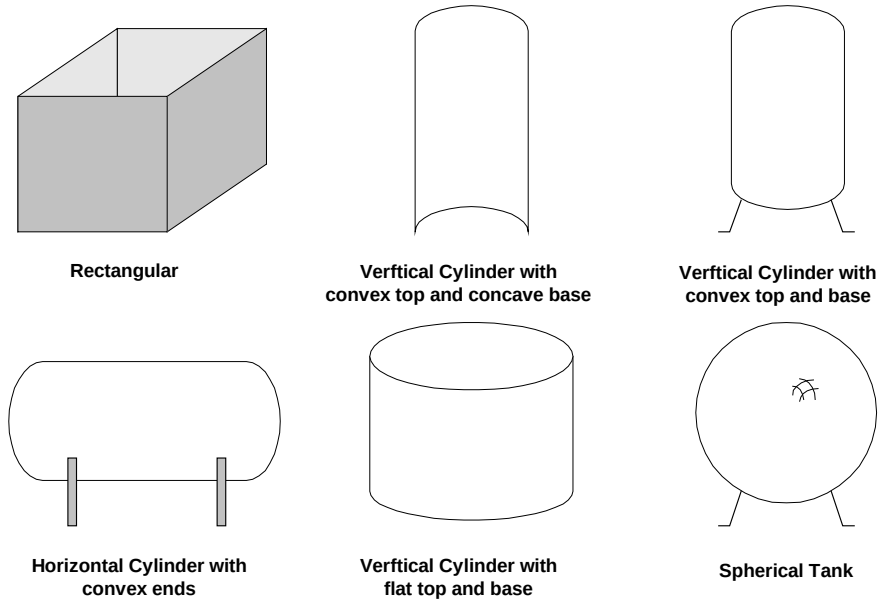


Figure 90: Types of tank shape

This shape is defined as a property *PatternType* defined as an enumeration within the property set *Pset_TankTypePreformedTank* e.g.

Property	Property Type	Value
PatternType	Enumerated Value	RECTANGULAR

Table 3: Setting the value for tank shape

Additionally, values are defined that enable the curvature of the ends of a horizontal or vertical cylinder. This allows specification of whether the ends are concave, convex or flat (or whether, as in the case of a spherical tank, unset). Values are defined in pairs so that it is possible to specify concave/convex etc. In this case, it is necessary to define a convention that enables the identification of which end is which.

The convention applied is that the first value in the pair represents the base (in the case of a vertical cylinder) or the left hand side (in the case of a horizontal cylinder when viewed as a front elevation). This becomes relevant when specifying the radius of curvature of shaped ends (below).

Radii of curvature are found from the properties *FirstCurvatureRadius* and *SecondCurvatureRadius* defined in the property set *Pset_TankTypePreformedTank*. As identified from the convention, the 'First' value is base or left hand side whilst 'Second' is the top or right hand side.

The actual values given for radii of curvature must be positive (through specification of the use of *IfcPositiveLengthMeasure*). This means that the curvature directions are identified from the terms 'concave' (for inward curvature) and 'convex' (for outward curvature.

Property	Property Type	Allowed Values
EndShapeType	Enumerated Value	CONCAVE/CONVEX
		FLAT/CONVEX
		CONVEX/CONVEX
		CONCAVE/FLAT
		FLAT/FLAT

Table 4: Setting values for curved end shapes

6.3.3.2.2 Sectional Tanks

A rectangular tank may be preformed or it may be sectional. Preformed tanks are generally relatively small with the body of the tank it is delivered to site in one piece. Sectional tanks tend to be large and are constructed on site from plate sections that are bolted together.

In the case of a sectional tank being specified, the property set *Pset_TankTypeSectionalTank* is also used to specify the number of sections to be used together with the length and width of each section. It should be noted that, in a sectional tank, the overall size is given by *Pset_TankTypeCommon* and the section sizes by *Pset_TankTypeSectionalTank*. There is no automatic cross checking that the overall size matches the tank size that would be determined by adding together the sizes of sections. It is up to the user, or the application employed by the user, to reconcile these sizes.

6.3.3.3 Property Sets Associated With a Tank Occurrence

Individual occurrences of a tank can also have properties. These are attached to the flow storage device occurrence however and not to the tank type

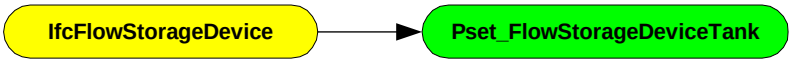


Figure 91: Property set associated with a tank occurrence

Property	***Property Type***	***Value***
TankComposition	Enumerated Value	ELEMENT
HasLadder	Enumerated Value	FALSE
HasVisualIndicator	Single Value	FALSE

Table 5: Setting values in the tank occurrence property set

```
/* definition of the type -no geometry shown- */
#1= IFCTANKTYPE('abcdefghijklmnopqrst11', #2, 'Tank_1', $, 'IfcFlowStorageDevice',
(#100,#200), (#1011), $, $, .PREFORMED.);

/* definition of the occurrence  -no geometry shown- */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $, (#20), #1);
#20=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst20', #2, $, $, $, #10001, #10002, $);

/* relating a property set to the occurrence */
#50=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst50', #2, $, $, (#20), #300);

/* definition of the common property set */
#100=IFCPROPERTYSET('abcdefghijklmnopqrst100', #2, 'Pset_TankTypeCommon', $,
(#110,#111,#112,#113,#114,#115,#116,#117,#118,#119));

/* definition of the properties within the common property set */
#110=IFCPROPERTYENUMERATEDVALUE('Type', $, (‘WATERSTORAGEPOTABLE’), #151);
#111=IFCPROPERTYENUMERATEDVALUE('AccessType', $, (‘LOOSECOVER’), #152);
#112=IFCPROPERTYSINGLEVALUE('NominalLengthOrDiameter', $, IFCLENGTHMEASURE(3), #99920);
#113=IFCPROPERTYSINGLEVALUE('NominalWidthOrDiameter', $, IFCLENGTHMEASURE(2), #99920);
#114=IFCPROPERTYSINGLEVALUE('NominalDepth', $, IFCLENGTHMEASURE(2), #99920);
#115=IFCPROPERTYSINGLEVALUE('NominalCapacity', $, IFCVOLUMEMEASURE(6), #99910);
#116=IFCPROPERTYSINGLEVALUE('EffectiveCapacity', $, IFCVOLUMEMEASURE(4.6), #99910);
#117=IFCPROPERTYSINGLEVALUE('Operating Weight', $, IFCMASSMEASURE(4950), #99900);
#118= IFCPROPERTYREFERENCEVALUE(‘Material’, $, $, #161);
#119=IFCPROPERTYSINGLEVALUE('MaterialThickness', $, IFCLENGTHMEASURE(.002), #99920);

/* definition of property enumerations used in common property set */
#151=IFCPROPERTYENUMERATION(‘PEnum_TankType’, (‘BREAKPRESSURE’, ‘EXPANSION’,
‘FEEDANDEXPANSION’, ‘GASSTORAGEBUTANE’, ‘GASSTORAGELIQUIFIEDPETROLEUMGAS’,
‘GASSTORAGEPROPANE’, ‘OILSERVICE’, ‘OILSTORAGE’, ‘WATERSTORAGEGENERAL’,
‘WATERSTORAGEPOTABLE’, ‘WATERSTORAGEPROCESS’, ‘WATERSTORAGECOOLINGTOWERMAKEUP’, ‘OTHER’,
‘NOTKNOWN’, ‘UNSET’), $);
#152=IFCPROPERTYENUMERATION(‘PEnum_TankAccessType’, (‘NONE’, ‘LOOSECOVER’, ‘MANHOLE’,
‘SECUREDCOVER’, ‘SECUREDCOVERWITHMANHOLE’, ‘OTHER’, ‘NOTKNOWN’, ‘UNSET’), $);

/* definition of property references used in common property set */
#161=IFCMATERIAL(‘Steel’);

/* definition of specific type property set */
#200=IFCPROPERTYSET('abcdefghijklmnopqrst200', #2, 'Pset_TankTypePreformedTank’, $,
(#210));

/* definition of specific properties for a preformed tank */
#210=IFCPROPERTYENUMERATEDVALUE('Type', $, (‘RECTANGULAR’), #251);
```

```
/* definition of property enumerations used in specific type property set */
#251=IFCPROPERTYENUMERATION(‘PEnum_TankPatternType’, (‘HORIZONTALCYLINDER’,
‘VERTICALCYLINDER’, ‘RECTANGULAR’, ‘OTHER’, ‘NOTKNOWN’, ‘UNSET’), $);

/* definition of occurrence property set */
#300=IFCPROPERTYSET('abcdefghijklmnopqrst300', #2, 'Pset_FlowStorageDeviceTank’, $,
(#310,#311,#312));

/* definition of specific properties for a preformed tank */
#310=IFCPROPERTYENUMERATEDVALUE('TankComposition', $, (‘ELEMENT’), #351);
#311=IFCPROPERTYSINGLEVALUE('HasLadder', $, IFCBOOLEAN(.F.), $);
#312=IFCPROPERTYSINGLEVALUE('HasVisualIndicator', $, IFCBOOLEAN(.F.), $);

/* definition of property enumerations used in specific type property set */
#351=IFCPROPERTYENUMERATION(‘PEnum_TankComposition’, (‘COMPLEX’, ‘ELEMENT’, ‘PARTIAL’,
‘NOTKNOWN’, ‘UNSET’), $);

/* definition of units used */
#99900=IFCSIUNIT(*,.MASSUNIT.,.KILO.,.GRAM.);
#99910=IFCSIUNIT(*,.VOLUMEUNIT.,$,.METRE.);
#99920=IFCSIUNIT(*,.LENGTHUNIT.,$,.METRE.);
```

6.3.3.3.1 Compartmentalized Tanks

Tanks may be provided so that fluid storage is in one compartment (the whole tank) or in multiple, separated compartments. The latter will be particularly the case with large tanks. This predominantly relates to rectangular tank constructions but could also relate to other shapes. The situation is similar to that of spatial structure elements where an element can be described as complex (multiple units), element (single unit) or partial (part of a unit).

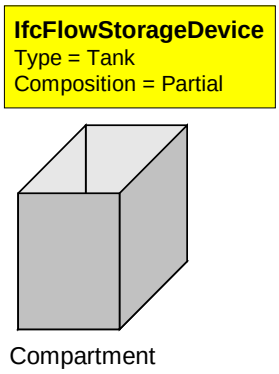


Figure 92: An individual tank compartment

The Solution can be determined by:

- Applying the composition principle to tanks.
- A compartment is an *IfcFlowStorageDevice* specified as a partial element through the occurrence level property set *Pset_FlowStorageDeviceTank* by setting the *TankComposition* value to PARTIAL
- A tank is an *IfcFlowStorageDevice* specified as a complete element through the occurrence level property set *Pset_FlowStorageDeviceTank* by setting the *TankComposition* value to ELEMENT
- Multiple tanks acting together through interconnection are an instance of *IfcTankType* specified as a complex element through the occurrence level property set *Pset_FlowStorageDeviceTank* by setting the *TankComposition* value to COMPLEX
- Use *IfcRelNests* to connect compartments to tanks (since both are instances of *IfcTank*)

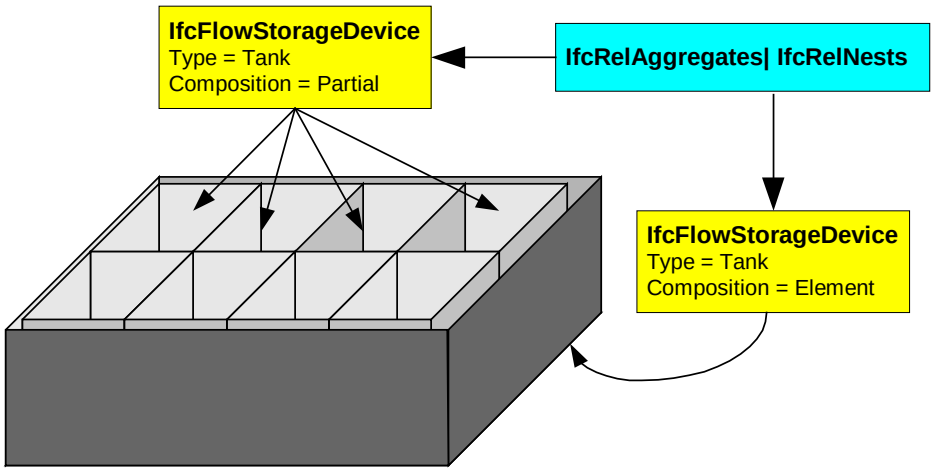


Figure 93: Aggregating multiple compartments into a single tank

```
/* TYPE LEVEL PROPERTY SETS NOT SHOWN IN THIS EXAMPLE */

/* definition of individual compartments by tank type -no geometry shown- */
#1= IFCTANKTYPE('abcdefghijklmnopqrst11', #2, 'Tank_Part', $, 'IfcFlowStorageDevice',
(#100,#200), (#1011), $, $, .PREFORMED.);

/* relating the occurrences to the type */
#10=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst10', #2, $, $,
(#20,#21,#22,#23,#24,#25,#26,#27), #1);

/* definition of the occurrences  - placement references given but no geometry shown- */
#20=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst20',#2,$,$,$,#10020,#10002,$);
#21=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst21',#2,$,$,$,#10021,#10002,$);
#22=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst22',#2,$,$,$,#10022,#10002,$);
#23=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst23',#2,$,$,$,#10023,#10002,$);
#24=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst24',#2,$,$,$,#10024,#10002,$);
#25=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst25',#2,$,$,$,#10025,#10002,$);
#26=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst26',#2,$,$,$,#10026,#10002,$);
#27=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst27',#2,$,$,$,#10027,#10002,$);
#28=IFCFLOWSTORAGEDEVICE('abcdefghijklmnopqrst28',#2,$,$,$,#10028,#10003,$);

/* placing the partial tanks into the elemental tank */
#30=IFCRELNESTS('abcdefghijklmnopqrst30', #2, $, ‘Compartmentalized tank nesting’, #28,
(#20,#21,#22,#23,#24,#25,#26,#27));

/* relating a property set to the occurrences */
#50=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst50', #2, $, $,
(#20,#21,#22,#23,#24,#25,#26,#27), #300);
#51=IFCRELDEFINESBYPROPERTIES('abcdefghijklmnopqrst51', #2, $, $, (#28), #301);

/* definition of occurrence property sets */
#300=IFCPROPERTYSET('abcdefghijklmnopqrst300', #2, 'Pset_FlowStorageDeviceTank’, $,
(#310,#321,#322));
#301=IFCPROPERTYSET('abcdefghijklmnopqrst301', #2, 'Pset_FlowStorageDeviceTank’, $,
(#311,#321,#322));

/* definition of specific properties for a preformed tank */
#310=IFCPROPERTYENUMERATEDVALUE('TankComposition', $, (‘PARTIAL’), #351);
#311=IFCPROPERTYENUMERATEDVALUE('TankComposition', $, (‘ELEMENT’), #351);
#321=IFCPROPERTYSINGLEVALUE('HasLadder', $, IFCBOOLEAN(.F.), $);
#322=IFCPROPERTYSINGLEVALUE('HasVisualIndicator', $, IFCBOOLEAN(.F.), $);

/* definition of property enumerations used in specific type property set */
#351=IFCPROPERTYENUMERATION(‘PEnum_TankComposition’, (‘COMPLEX’, ‘ELEMENT’, ‘PARTIAL’,
‘NOTKNOWN’, ‘UNSET’), $);
```

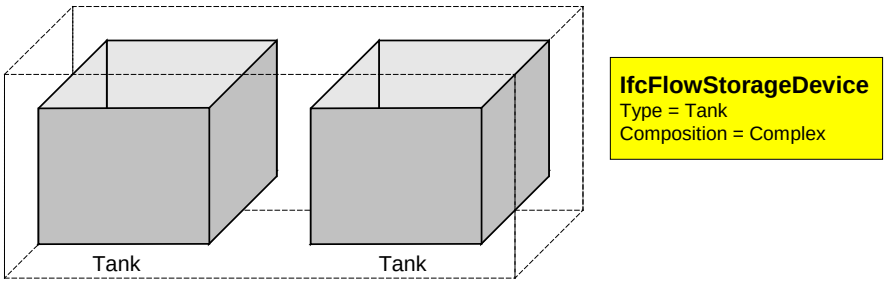
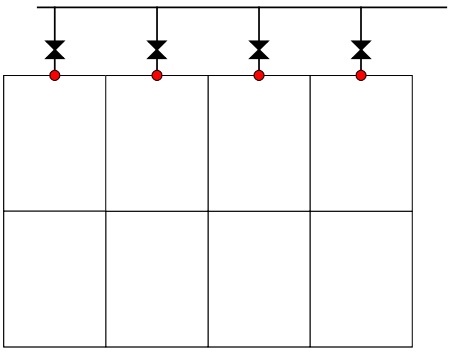


Figure 94: Aggregating individual tanks into a tank complex

Each compartment of the tank can be recognized with its own properties instead of trying to capture these within an overall tank property set. A tank with compartments of different sizes can be composed (instead of assuming that all compartments are the same size) and separate tanks acting together to fulfil a common role can be identified as a tank complex.



Compartmentalised Tank with Connections
(showing ports on compartments/partial tanks)

Figure 95: Connections from tank compartments

6.4 Building service system engineering concepts

The concepts of system and zoning for building service elements are important to building services specific view definitions of IFC2x2 – however they don't form part of the basic coordination view.

6.4.1 Concept of system

A system is defined as an 'organized combination of related parts within an AEC product, composed for a common purpose or function or to provide a service'. The key concepts of this definition are that the parts are related and that, in composition, they fulfill a common purpose.

Within the IFC Model, a system is defined as a subtype of *IfcGroup*. That is, it acts as a functional related aggregation of objects. However, whilst the *IfcGroup* can aggregate any set of instances that are subtypes of *IfcObject*, the *IfcSystem* is restricted by a WHERE rule so that it can only aggregate subtypes of *IfcElement* or other (sub-)systems.

```
ENTITY IfcSystem
   SUBTYPE OF (IfcGroup);
   INVERSE
      ServicesBuildings : SET [0:1] OF IfcRelServicesBuildings FOR RelatingSystem;
   WHERE
      WR1 : SIZEOF (QUERY (temp <* SELF\IfcGroup.IsGroupedBy.RelatedObjects |
         NOT (('IFCPRODUCTEXTENSION.IFCELEMENT' IN TYPEOF(temp)) OR
              ('IFCPRODUCTEXTENSION.IFCSYSTEM' IN TYPEOF(temp)))
         )) = 0;
END_ENTITY;
```

Whilst this allows for any subtype of element to participate in an *IfcSystem*, the definition that they should be related is significant. Typically, an *IfcSystem* will act for a particular purpose and the elements that are aggregated into the system should be consistent withy that purpose. For instance, in an architectural context, a system might be used to define a functionally related set of walls. In structural engineering, this might be beams and columns whilst in a building services context, a

system will comprise distribution elements (pipes or ducts or cables and related items). Objects that are clearly intended for use within one purpose should not be mixed in a system with objects that are clearly intended for a different purpose.

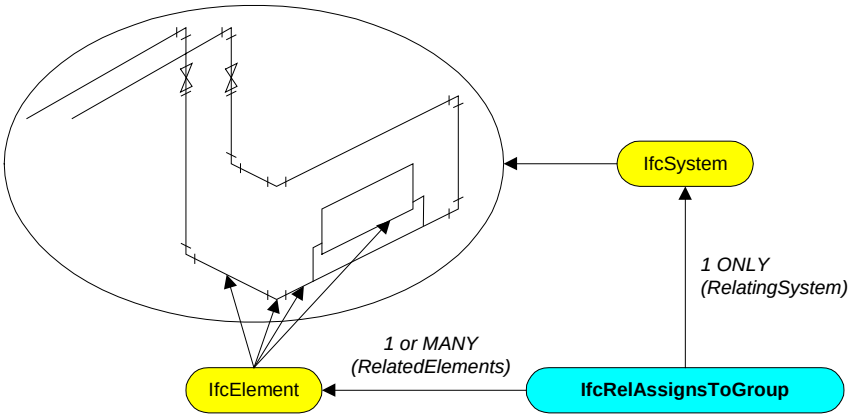


Figure 96 : Elements aggregated into a system

In general, it is expected that the elements that comprise a system will be connected together in a 'systematic' way. However, there is no enforcement of a connectivity requirement within *IfcSystem*[14]. Therefore, if there is a need for elements to be connected (e.g. to establish a flow path), this has to be defined prior to an IFC export or be determined by applying external rules procedures to the exported information.

6.4.1.1 System Naming

There may be many instances of *IfcSystem* within a building, each system fulfilling a different purpose. To distinguish between them, it is important that each system is named or identified according to what it does. There are two potential strategies that can be used for system naming. These are explicit naming and classification.

For explicit naming, systems are identified using the inherited *Name* attribute from *IfcRoot*. This resolves to a simple STRING data type. Note also that the attribute *ObjectType* inherited from *IfcObject* may be optionally used to qualify the system name further. The latter is particularly used together with property set or type definitions attached to the *IfcSystem*.

An enumeration of system types is not used, firstly because of the enormous range of system types that can exist and secondly because naming of system types is likely to vary from place to place. Where selection is to be made from a constrained list, it is strongly recommended that the classification strategy is used and that a list of system types within a published classification is used.

For detailed information on implementation of classification within IFC, a reference is made to the relevant section 10.4.2. An example of classified system types extracted from the UNICLASS[15] specification is given below. In UNICLASS, systems are identified within sections G5 (Mechanical) and G6 (Electrical):

System Name	Code		
Cold water systems	G501	Lifts	G561
Hot water supply systems	G502	Escalators	G562
Gas supply systems	G51	Conveyors	G563
Heating systems	G52	Traveling cradle systems	G564
HVAC	G52	Entrance controls	G5711
Air conditioning systems	G52	Security alarm systems	G5712
Electric power	G53	Smoke detection/alarm systems	G5721
Power	G53	Fire detection and alarm systems	G5721
Lighting	G54	Sprinkler systems	G5722

[14] Up to IFC 2x2 Addendum 1

[15] UNICLASS is a facetted classification system based on the classification systems framework of ISO 12006 Part 2 and developed for use within UK practice.

System Name	Code		
Public address systems	G551	Fire fighting systems	G5722
Visual display communications	G552	Fire suppression systems	G5722
Radio communication systems	G553	Smoke extract/control systems	G5723
TV communication systems	G554	Lightning protection	G5731
Telecommunications systems	G555	Drainage	G581
Computer network communications	G556	Foul drainage	G5811
Hoists	G561	Surface water drainage services	G5812
		Refuse chute disposal systems	G582
		Monitoring/control systems	G66

Table 6 : Example of UNICLASS system definitions

6.4.1.2 System Naming in Applications

There are many cases where system names provided through an IFC exchange need to comply with particular system types. Examples include:

- application of varying design parameters in different system types;
- checking that a system or parts of a system comply with the requirements of building codes.

It is likely that agreements on system naming will need to be made according to local or project usage or through use of a named classification system. In either case, it is important that the application exports a system name or identity that can be understood by the downstream (receiving) application according to the defined criteria.

6.4.1.3 Multifunctionality

Some elements can participate in more than one group (generically) or more than one system (specifically). The ability of an element to participate in more than one performing group is termed multifunctionality. Multifunctionality is achieved by assigning an element to more than one instance of *IfcGroup* (or its subtype *IfcSystem*) through the definition of multiple instances of *IfcRelAssignsToGroup*.

For instance, a pipe may belong to one system group for fabrication purposes, a second system group for building code checking purposes and a third group for scheduling purposes. Each group in this case may be defined as a subsystem (see below), the element still participates in only one overall system.

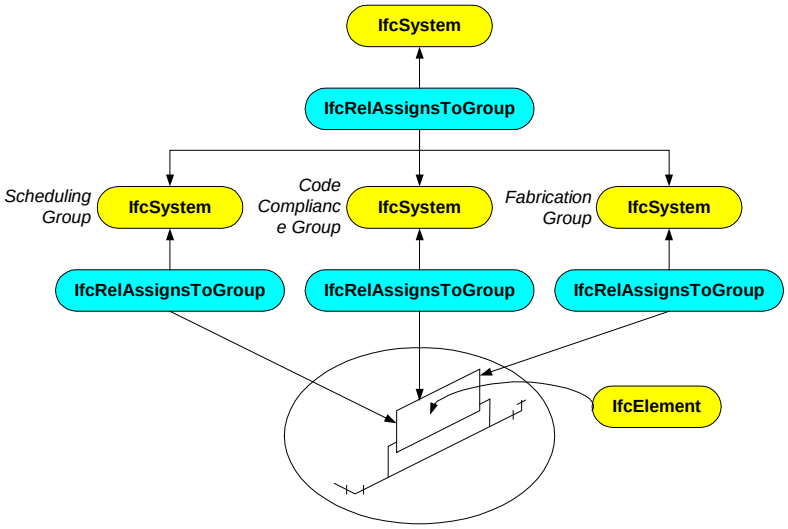


Figure 97 : Multifunctional element in multiple subsystems

Alternatively, an element may be performing a distribution role in one system context but a structural role in a different system context. In this case, the element is assigned to two completely separate systems through two instances of *IfcRelAssignsToGroup*.

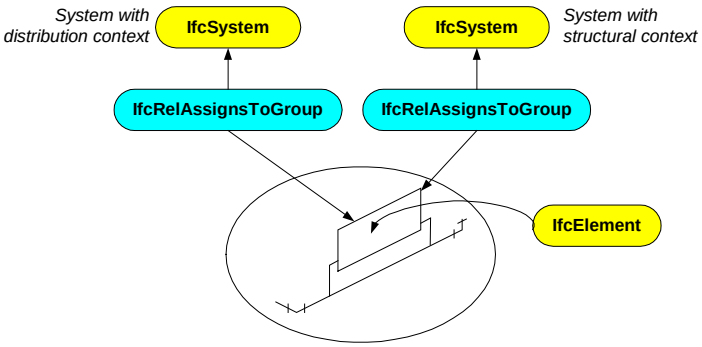


Figure 98 : Multifunctional element in multiple system contexts

6.4.1.4 Subsystems

There is often a need to be able to identify related subsets of elements within a system. Subsets, or subsystems, may be required for many purposes including for off-site fabrication, identification for building code compliance, construction scheduling (and 4D animation).

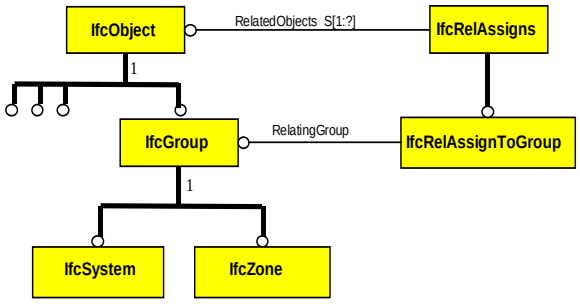


Figure 99 : A group can contain other groups

Within IFC 2x2, an *IfcGroup* (the supertype of *IfcSystem*) is specified as a subtype of *IfcObject*. Other objects are related to that group through *IfcRelAssignsToGroup*. The *IfcRelAssigns* class (which is the supertype of *IfcRelAssignsToGroup*) is used to provide the set of related instances of *IfcObject*. Because it is a subtype of *IfcObject*, an instance of *IfcGroup* can participate as a member of the set of objects forming another group.

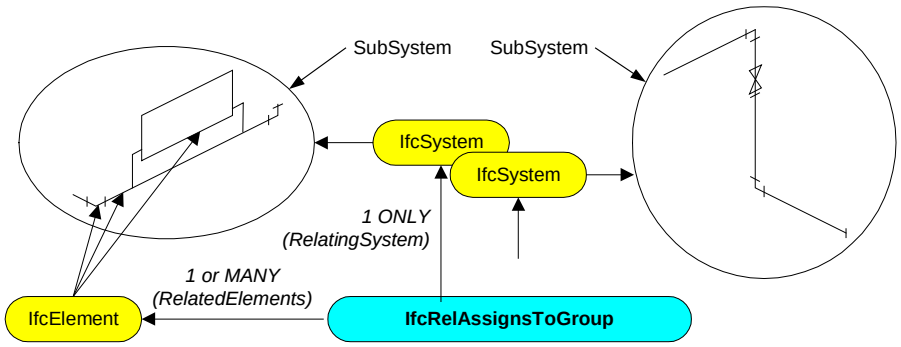


Figure 100 : Elements aggregated into subsystems

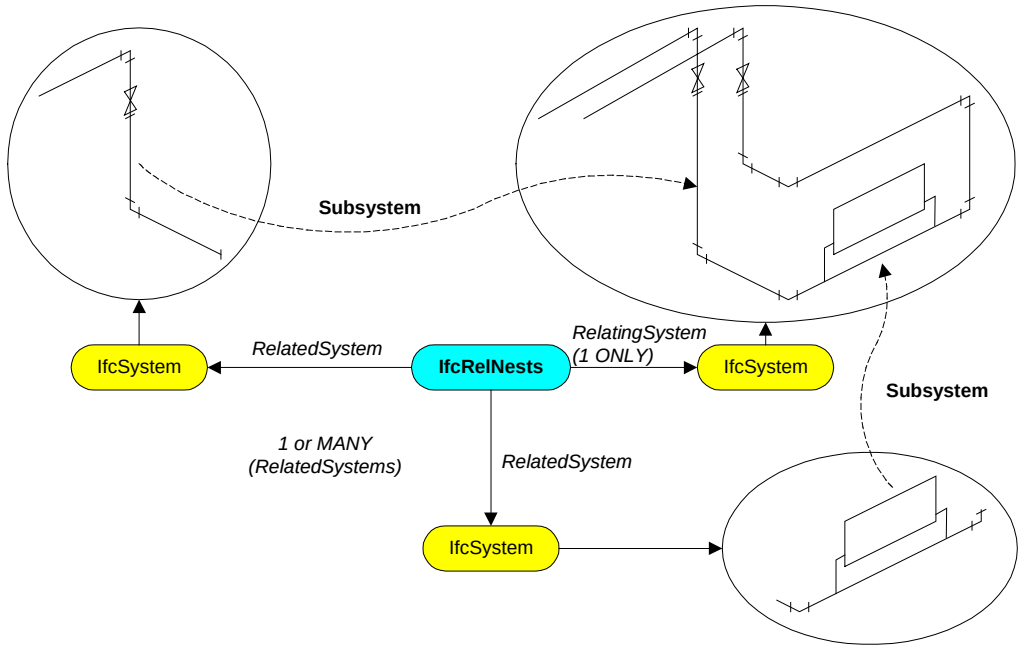


Figure 101 : Systems and subsystems with hierarchical relations

A nesting relationship is considered to exist between a system and a subsystem (as opposed to the grouping relationship that exists between a system and its contained elements). Within IFC 2x2, a nesting relationship is defined as a decomposition of objects of the same type. That is, an *IfcSystem* can also nest other instances of *IfcSystem*.

```
/* Element definitions */
#1001=IFCFLOWSEGMENT ('abcdefghijklmnopqrs113', #1, $, $, $, $, $, $);
#1002=IFCFLOWFITTING ('abcdefghijklmnopqrs213', #1, $, $, $, $, $, $);
#1003=IFCFLOWSEGMENT ('abcdefghijklmnopqrs313', #1, $, $, $, $, $, $);
#1004=IFCFLOWFITTING ('abcdefghijklmnopqrs413', #1, $, $, $, $, $, $);
#1005=IFCFLOWFITTING ('abcdefghijklmnopqrs513', #1, $, $, $, $, $, $);
#1006=IFCFLOWFITTING ('abcdefghijklmnopqrs613', #1, $, $, $, $, $, $);
#1007=IFCFLOWSEGMENT ('abcdefghijklmnopqrs713', #1, $, $, $, $, $, $);
#1008=IFCFLOWFITTING ('abcdefghijklmnopqrs813', #1, $, $, $, $, $, $);
#1009=IFCFLOWSEGMENT ('abcdefghijklmnopqrs913', #1, $, $, $, $, $, $);
#1010=IFCFLOWFITTING ('abcdefghijklmnopqr1013', #1, $, $, $, $, $, $);
#1011=IFCFLOWSEGMENT ('abcdefghijklmnopqr1113', #1, $, $, $, $, $, $);
#1012=IFCFLOWFITTING ('abcdefghijklmnopqr1213', #1, $, $, $, $, $, $);
#1013=IFCFLOWTERMINAL('abcdefghijklmnopq13t13', #1, $, $, $, $, $, $);
#1014=IFCFLOWFITTING ('abcdefghijklmnopqr1413', #1, $, $, $, $, $, $);
#1015=IFCFLOWSEGMENT ('abcdefghijklmnopqr1513', #1, $, $, $, $, $, $);

/* System definitions */
#2001=IFCSYSTEM ('abcdefghijklmnopqrst02',#2,'HeatingSystem', $,'ParentSystem');
#2002=IFCSYSTEM ('abcdefghijklmnopqrst03',#3,'DropPipeSubsystem', $,'Subsystem');
#2003=IFCSYSTEM ('abcdefghijklmnopqrst04',#4,'EmitterSubsystem', $,'Subsystem');

/* Assignment of Elements to Systems */
#3001=IFCRELASSIGNSTOGROUP
('abcdefghijklmnopqrst52',#5,$,$,(#1001,#1002,#1003,#1004,#1005,#1006,#1007,#1008,#1009,
#1010,#1011,#1012,#1013,#1014,#1015),.PRODUCT.,#2001);
#3002=IFCRELASSIGNSTOGROUP
('abcdefghijklmnopqrst53',#6,$,$,(#1001,#1002,#1003,#1004,#1005,#1006,#1007,#1008),.PROD
UCT.,#2002);
#3003=IFCRELASSIGNSTOGROUP
('abcdefghijklmnopqrst54',#7,$,$,(#1009,#1010,#1011,#1012,#1013,#1014,#1015),.PRODUCT.,#
2003);

/* Nesting of subsystems */
#4001=IFCRELNESTS ('abcdefghijklmnopqrst62',#8,$,$,#2001,(#2002,#2003));
```

7 Structural Elements

8 Annotations

9 Shape Representation of Elements

9.1 Geometric representation of products

This chapter introduces the concept of geometric representations of products within the IFC2x model. Each leaf node object in IFC that derives from *IfcProduct* can have geometric representations. The schemas holding the resource definitions are:

- *IfcGeometryResource* - Defines the basic geometric representation items.
- *IfcTopologyResource* - Defines the basic topological representation items.
- *IfcGeometricModelResource* - Defines the geometric models to define the geometric representation of objects. It utilizes the standard geometric and topological representation items to represent geometric models.
- *IfcRepresentationResource* - Defines the concepts of multiple shape representation of objects and the reference to a representation context.
- *IfcGeometryConstraintResource* - Defines the concept of object placement, either using absolute or relative axis placement, or by referring to a placement relative to grid definitions.
- *IfcProfileResource* - Defines standard profile (or cross section) definitions to be used in geometric representations or models of swept surfaces or swept solids.

The first three resources incorporate definitions and concepts taken from the international standard ISO/IS 10303-42 "Integrated generic resources: Geometric and topological representations". The fourth resource incorporates definitions and concepts taken from the international standard ISO/IS 10303-43 " Integrated Generic Resources – Representation Structures". The fifth and sixth resource are true additions defined within the IFC model that do not have current counterparts within the Integrated Resources as provided under ISO 10303.

In IFC2x2 the concept of visual presentations of geometric representation items has been added to the IFC model. The presentation capabilities include definition of presentation style attributes for realistic and symbolic visualizations of geometric element, including presentation styles for curve, text, filled area, symbol, and surface. In addition it supports the definition of pre defined or externally defined character fonts, symbols, colors, and line types. Also the image control by a layer mechanism is now in scope. The presentation capabilities have two purposes:

- to add the explicit style information for the shape representation of products
- to add additional (no directly product related) annotations within the same model space as the product shape representations

These new concepts will be explained in a future version of the IFC2x model implementation guide.

9.1.1 Reference to the geometric representation

Any object in IFC that has a geometric representation (i.e. is a subtype of *IfcProduct*) needs to have a value for the two attributes, inherited from *IfcProduct*. Taking *IfcWall* :

In the IFC model it is defined as:

```
ENTITY IfcWall;
   ENTITY IfcRoot;
      GlobalId      :   IfcGloballyUniqueId;
      OwnerHistory  :   IfcOwnerHistory;
      Name          :   OPTIONAL IfcLabel;
      Description   :   OPTIONAL IfcText;
   ENTITY IfcObject;
      ObjectType    :   OPTIONAL IfcLabel;
    INVERSE
         (...)
   ENTITY IfcProduct;
      ObjectPlacement  :   OPTIONAL IfcObjectPlacement;
      Representation   :   OPTIONAL IfcProductRepresentation;
    INVERSE
         (...)
   ENTITY IfcElement;
         (...)
   ENTITY IfcBuildingElement;
```

```
        (...)
END_ENTITY;  --IfcWall
```

And in the IFC file it is exchanged as:

```
#1=IFCWALL('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);
#3=IFCLOCALPLACEMENT($, #10);
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11));
```

If a *Representation* is given to the subtype of *IfcProduct*, the *ObjectPlacement* has to be inserted as well. This constraint is enforced by a local rule at *IfcProduct*.

The two attributes are described below. They are defined at the level of *IfcProduct* and equally apply to all subtypes. Therefore a single concept of describing the geometric representation is applied throughout the whole IFC schema, and therefore the method of geometric representation is the same for e.g. *IfcSpace, IfcWall, IfcFlowSegment, IfcFurniture, IfcOpeningElement, IfcProxy*.

- *ObjectPlacement* Placement of the product in the spatial context, the placement can either be absolute (relative to the world coordinate system), relative (relative to the object placement of another product), or constrained (e.g. relative to grid axes).
- *Representation* Association to the product representation, which is the container of potentially multiple geometric representations. All geometric representations of the product are defined within the *ObjectPlacement*, which provides the object coordinate system.

The next sections describe the concept of object placement, of multiple product representation, and of the different types of shape representation in further details.

All coordinates and geometric parameters (which are length or angle measures) are unit dependent. The units common to all geometric representations within an IFC exchange are defined globally, i.e. within the context of *IfcProject* (see 3.1). The UoF "Global Unit Assignment" is further described within 3.3.2).

9.1.2 Concept of object placement

Any product defined as subtype of *IfcProduct* in IFC can have one or more geometric representations (e.g. a simple representation as a bounding box, and/or a complex representation as a boundary representation model). All of the geometric representations of the same object are defined within the same object coordinate system. This object coordinate system is given by the reference *ObjectPlacement* to *IfcObjectPlacement*.

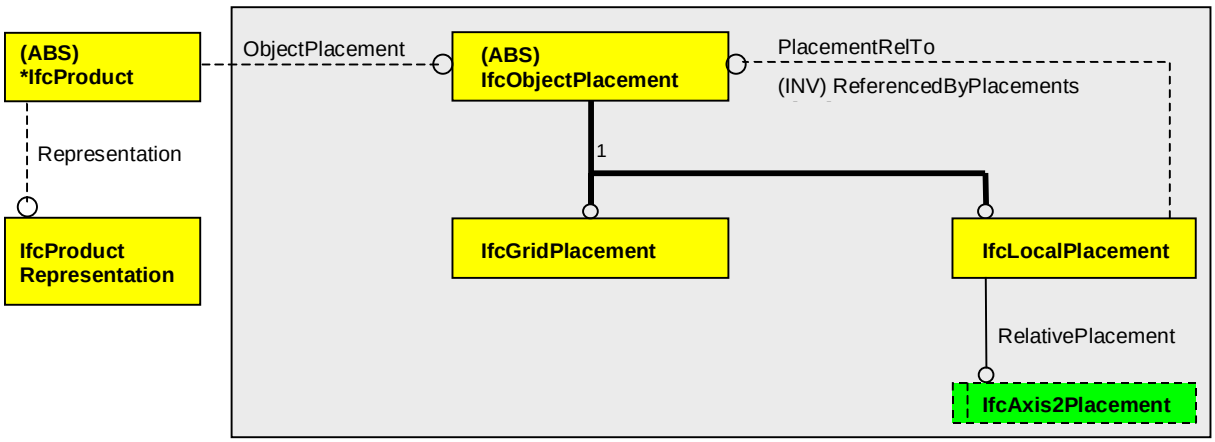


Figure 102 : Definition of IfcObjectPlacement

The *IfcObjectPlacement* has to be provided for each product, which has a shape representation defined. The entity *IfcObjectPlacement* is abstract and only the two subtypes, *IfcLocalPlacement* and *IfcGridPlacement* can be instantiated.

The object placement can be given:

- absolute, i.e. by an axis2 placement, relative to the world coordinate system,

- relative, i.e. by an axis2 placement, relative to the object placement of another product,
- by grid reference, i.e. by the virtual intersection and reference direction given by two axes of a design grid.

The object placement defines unambiguously the object coordinate system as either two-dimensional axis placement (*IfcAxis2Placement2D*) or three-dimensional axis placement (*IfcAxis2Placement3D*). The axis placement may have to be calculated, e.g. in the case of a grid placement.

9.1.2.1 Concept of local placement

The default way to define the placement of a product is using the relative placement, given by *IfcLocalPlacement* with an inserted value for the *PlacementRelTo* attribute. If the value of *PlacementRelTo* is omitted, then the placement is given in global coordinates, i.e. within the global coordinate system, as established at the *IfcGeometricRepresentationContext*.

9.1.2.1.1 Concept of local relative placement

The *IfcLocalPlacement* with the *PlacementRelTo* attribute value inserted defines the relative placement of a product in relation to the placement of another product and finally through the intermediate referenced placements within the geometric representation context of the project.

Each local placement is given by an axis placement, which can be either a 2D or a 3D axis placement. The type of the axis placement shall be the same for the local relative placement and the referenced placement. The axis placement, of type *IfcAxis2Placement2D* or *IfcAxis2Placement3D,* is given by:

- the location
- the direction of the local Z-axis (in case of 3D) – if omitted always [0.,0.,1.]
- the direction within the positive XZ plane (in case of 3D) or the direction of the X-axis (in case of 2D) – if omitted always [1.,0.,0.] – or [1.,0.] for 2D

Example:

In the following example, several local placements are used, that are defined relatively. Only the placement objects, IfcLocalPlacement, are shown, but not the subtypes of IfcProduct, which utilizes the placement objects. The IfcGeometricRepresentationContext is given as it establishes the global (or world) coordinate system.

```
/* Definition of the world coordinate system */
#1=IFCGEOMETRICREPRESENTATIONCONTEXT($, '3Dmodel', 3, 1.0E-005, #2, $);
#2=IFCAXIS2PLACEMENT3D(#3, $, $);
#3=IFCCARTESIANPOINT((0.0, 0.0, 0.0));

/* Definition of the local coordinate systems */
#4=IFCLOCALPLACEMENT($, #7);
#7=IFCAXIS2PLACEMENT3D(#10, $, $);
#10=IFCCARTESIANPOINT((0.0, 0.0, 2.0));

#5=IFCLOCALPLACEMENT(#4, #8);
#8=IFCAXIS2PLACEMENT3D(#11, $, $);
#11=IFCCARTESIANPOINT((1.0, 0.0, 0.0));

#6=IFCLOCALPLACEMENT(#4, #9);
#9=IFCAXIS2PLACEMENT3D(#12, #13, #14);
#12=IFCCARTESIANPOINT((0.0, 2.0, 2.0.));
#13=IFCDIRECTION((0.0, 1.0, 0.0.));
#14=IFCDIRECTION((0.0, 0.0, -1.0));
```

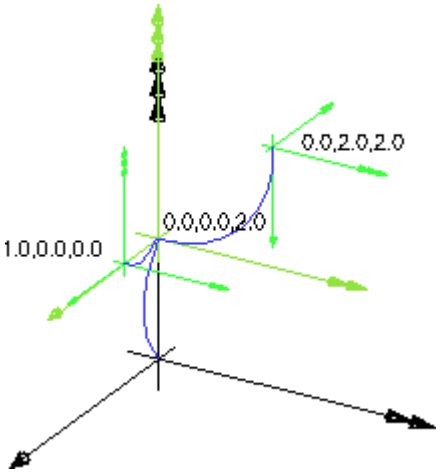


Figure 103 : Example of local relative placement

There are several rules applied to the referencing of parent placements. The referencing has to be a-cyclic and the following shall apply to the subtypes of *IfcProduct*, which have a placement:

- *IfcSite* shall be placed absolutely within the world coordinate system established by the geometric representation context of the project
- *IfcBuilding* shall be placed relative to the local placement of *IfcSite*
- *IfcBuildingStorey* shall be placed relative to the local placement of *IfcBuilding*
- *IfcElement* shall be placed relative:
  - to the local placement of its container (normally to *IfcBuildingStorey* but possibly also to *IfcSite*, *IfcBuilding*,), or
  - to the local placement of the *IfcElement* to which it is tied by an element relationship (i.e. by one of the following relations *IfcRelVoidsElement, IfcRelFillsElement, IfcRelCoversBldgElements, IfcRelAssembles*)

9.1.2.1.2 Concept of local absolute placement

The *IfcLocalPlacement* with the *PlacementRelTo* attribute left unassigned defines the absolute placement of a product in relation to the world coordinate system as established by the assigned geometric representation context of the project. The definition of the axis placement is the same as for the local relative placement (see 9.1.2.1.1).

Example:

Taking a similar example as for the local relative placement, several object placements are used, that are defined absolutely in regard to the world coordinate system. Only the placement objects, IfcLocalPlacement, are shown, but not the subtypes of IfcProduct, which utilizes the placement objects. The IfcGeometricRepresentationContext is given as it establishes the global (or world) coordinate system.

```
/* Definition of the world coordinate system */
#1=IFCGEOMETRICREPRESENTATIONCONTEXT($, '3Dmodel', 3, 1.0E-05, #2, $);
#2=IFCAXIS2PLACEMENT3D(#3, $, $);
#3=IFCCARTESIANPOINT((0.0, 0.0, 0.0));

/* Definition of the local absolute coordinate systems */
#4=IFCLOCALPLACEMENT($, #7);
#7=IFCAXIS2PLACEMENT3D(#10, $, $);
#10=IFCCARTESIANPOINT((3.8, 1.4, 0.0));

#5=IFCLOCALPLACEMENT($, #8);
#8=IFCAXIS2PLACEMENT3D(#11, #16, #17);
#11=IFCCARTESIANPOINT((1.2, 1.4, 0.0));
#16=IFCDIRECTION((0.0, 0.0, 1.0));
#17=IFCDIRECTION((0.96592, 0.25881, 0.0));
```

```
#6=IFCLOCALPLACEMENT($, #9);
#9=IFCAXIS2PLACEMENT3D(#12, #13, #14);
#12=IFCCARTESIANPOINT((2.2., 2.4, 5.0));
#13=IFCDIRECTION((1.0, 0.0, 0.0));
#14=IFCDIRECTION((0.0, 0.0, -1.0));
```

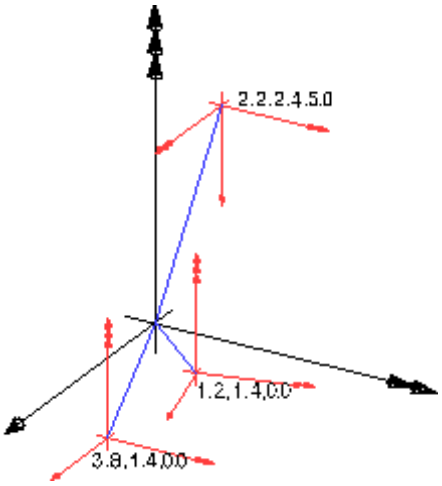


Figure 104 : Example of global absolute placement

9.1.2.2 Concept of placement relative to grid

The subtype *IfcGridPlacement* allows to definition of an object placement relative to the intersection of two grid lines, *IfcGridAxis*, within the same grid, *IfcGrid* through the *IfcVirtualGridIntersection* relation.

The *IfcVirtualGridIntersection* specifies the object placement of the object coordinate system by pointing to exactly two *IfcGridAxis*. The intersection point, established by the *PlacementLocation,* defines the *Location* of the object placement (which maybe offset from the exact intersection point). The X-axis orientation of the object placement defaults to the direction of the first intersection axis, if no *PlacementRefDirection* is given, otherwise it is computed by the vector between the *PlacementLocation* and the *PlacementRefDirection*.

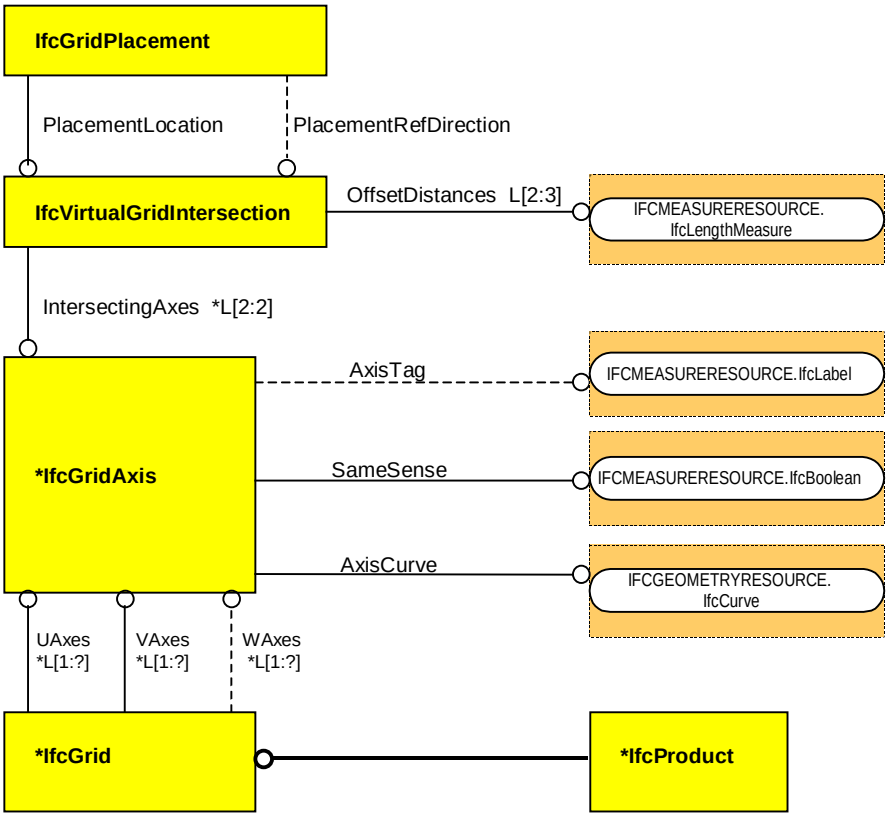


Figure 105 : Definition of placement relative to grid

The functionality supported by the *IfcGridPlacement* thereby includes:

- object placement relative to the intersection of two grid axes
- object placement relative to an offset to the intersection of two grid axes
- object placement relative to the intersection of two grid axes and orientation relative to the direction given by the second pair of grid axes
- object placement relative to an offset to the intersection of two grid axes and orientation relative to the direction given by the second pair of grid axes

Example:

*The placement of an element, e.g. a column (not shown in the *.ifc file), is given relative to the intersection of two grid axes. The placement is offset in the x and y-direction and gets its orientation from the tangent of the first grid axis. The grid axes are defined within the local placement of the grid.*

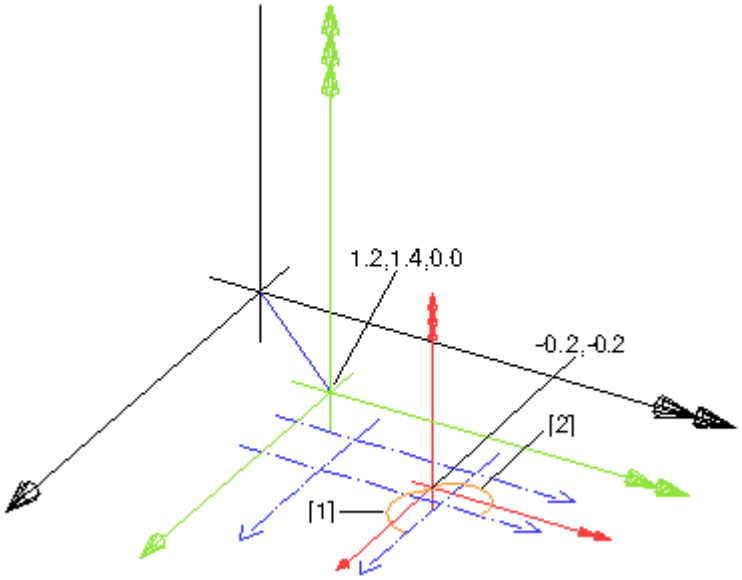


Figure 106 : Example for placement relative to grid

More explanations to the grid placement are later given at the description of the leaf node entity *IfcGrid* elsewhere in the document.

9.1.3 Concept of multiple product shape representations

Within the same object placement, a product may be represented by a single or multiple shape representations – this functionality is introduced as UoF "multiple shape representations". Each of the individual shape representations may refer to a different representation context and have a different representation name and type. However all geometric representation contexts are required to refer to the same global (or world) coordinate system.

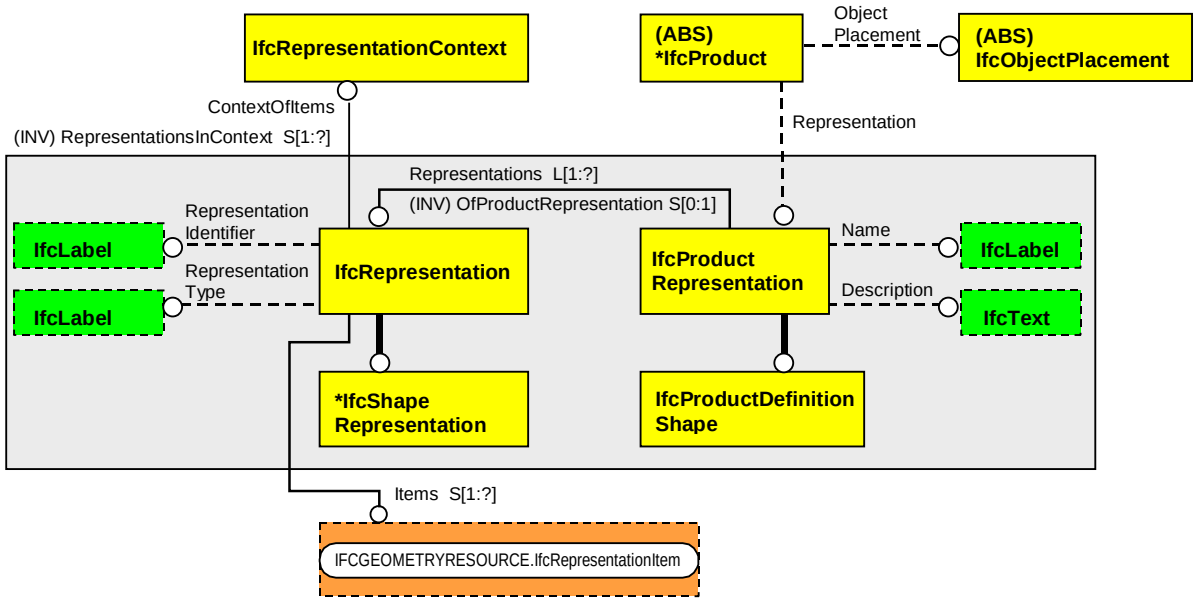


Figure 107 : Definition of (multiple) shape representations

If the product has a product representation, it also has to have an object placement. The object placement provides for the object coordinate system, in which all the geometric representation items of all shape representations are founded. For geometric representations (established by the use of *IfcGeometricRepresentationItem's* as items for single or multiple shape representations) the subtypes *IfcProductDefinitionShape* and *IfcShapeRepresentation* have to be used.

There are two concepts which have to be supported for single and multiple representations of product shape.

- *IfcProductRepresentation* (and subtype *IfcProductDefinitionShape*) as the container of all representations for a product (which are founded in the object placements, as discussed in 9.1.2)
- *IfcRepresentation* (and subtype *IfcShapeRepresentation*) as the container of all representation items, which are used for a particular representation (within the multiple representations possible) for a product. Each of the *IfcRepresentation*'s would be assigned to a *IfcRepresentationContext* (or subtype *IfcGeometricRepresentationContext*).

9.1.3.1 Concept of product representation and product definition shape

The *IfcProductRepresentation* (or the *IfcProductDefinitionShape* as required for geometric representations) represents the container for all individual, or alternative representations of the same product. It allows for a characterization of the product representation by a *Name* and for the provision of an additional *Description*. Currently there are no recommendations on how to use both attributes. However additional recommendations maybe be given within particular views or implementation agreements. In general it can be said that the *Name* may be used as a specific identifier for the product representation as a whole (as opposed to the name of individual representations contained within the instance). The *Description* may be used to add further meaningful information that characterizes the instance.

An *IfcProductRepresentation* (or the *IfcProductDefinitionShape*) is a property of *IfcProduct* (introduced at the *IfcKernel* level) that defines the geometric or topological representation of an instance of *IfcProduct*. Note that the only subtype of *IfcObject* that can have a product representation is *IfcProduct*.

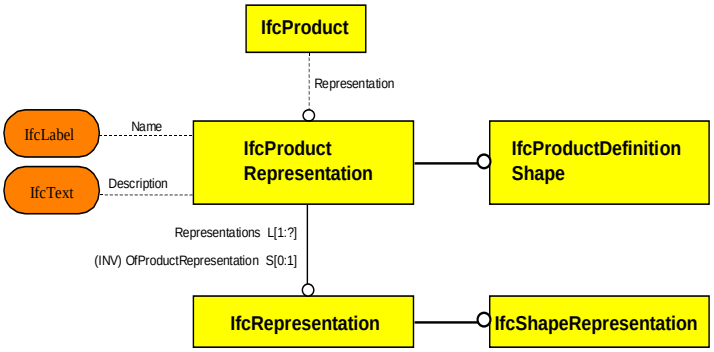


Figure 108 : Definition of product representation

An *IfcProduct* may have one or many representations. For instance, there may be a topological representation and any of several different types of geometric representation (see discussion on *IfcShapeRepresentation* below). An instance of *IfcProductRepresentation* (or subtype) acts as a container for all of the geometric and topological representations that exist for a product.

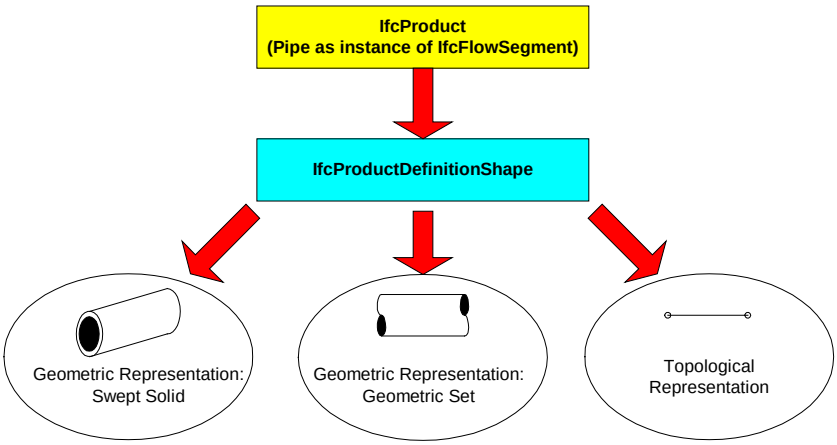


Figure 109 : Multiple representations of a product

9.1.3.2 Concept of shape representation

Each *IfcProductRepresentation* (or subtype *IfcProductDefinitionShape*) has to have at least one *IfcRepresentation* (or subtype *IfcShapeRepresentation*) defined, whereby the subtype *IfcProductDefinitionShape* is required to have at least one *IfcShapeRepresentation* assigned.

Each representation may carry additional classifications provided by the *RepresentationIdentifier* and the *RepresentationType*. The provision of the *RepresentationType* is required for the subtype *IfcShapeRepresentation*. The IFC2x model describes all allowed values for the *RepresentationType* and makes recommendations for the use of the *RepresentationIdentifier*. Therefore both attributes should only be used to carry attribute values are described within the IFC2x specification, mainly within the geometry use sections of the individual subtypes of *IfcElement*.

Example:

The shape representation of a standard wall shall include the wall path and the wall body. It would require a multiple representation of the IfcProductDefinitionShape by assigning two instances of IfcShapeRepresentation. The two attributes RepresentationIdentifier and RepresentationType have the following values:

IfcShapeRepresentation for the wall path:

- *RepresentationIdentifier* :: "Axis"
- *RepresentationType* :: "Curve2D"

IfcShapeRepresentation for the wall body:

- *RepresentationIdentifier* :: "Body"
- *RepresentationType* :: "SweptSolid"

The values for *RepresentationIdentifier* and *RepresentationType* have to be specified either within the IFC model specification or within the specific views for implementation. Implementations of the IFC Geometry capabilities should support these conventions.

Example:

*As discussed in the previous example the following picture and the attached *.ifc file cut-out shows the multiple geometric representation of a wall, including the wall axis and the wall basis. It should be noted, that the presentation (i.e. the line color (blue and red) and the line style (continuous and dashed)) is not currently in scope of the IFC2x based exchange. The display of the wall is determined by the wall style settings within the various software systems, participating in the IFC based exchange.*

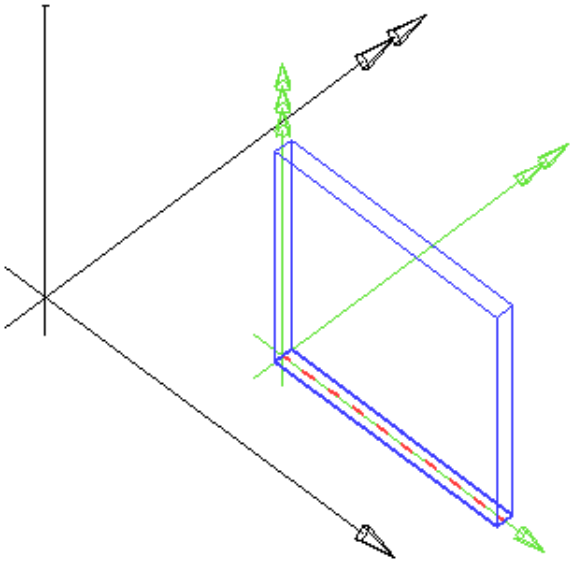


Figure 110 : Example of multiple geometric representation of wall

```
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);

/* UoF local absolute placement */
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.000000E+00, 1.000000E+00, 0.000000E+00));

/* UoF representation context */
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));

/* UoF multiple geometric representations */
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11, #13));

/* first geometric representation for the wall axis */
#11=IFCSHAPEREPRESENTATION(#12, 'Axis', 'Curve2D', (#18));
#18=IFCTRIMMEDCURVE(#19,(#20),(#21),.T.,.CARTESIAN.);
#19=IFCLINE(#30, #31);
#30=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#31=IFCVECTOR(#32,2.8E+00);
#32=IFCDIRECTION((1.0E+00, 0.0E+00));
#20=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#21=IFCCARTESIANPOINT((2.80E+00, 0.0E+00));

/* second geometric representation for the wall body */
#13=IFCSHAPEREPRESENTATION(#12, 'Body', 'SweptSolid', (#22));
#22=IFCEXTRUDEDAREASOLID(#23, #26, #29, 2.80E+00);
#26=IFCAXIS2PLACEMENT3D(#28, $, $);
#28=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));
#29=IFCDIRECTION((0.0E+00, 0.0E+00, 1.0E+00));
#23=IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #40);
#40=IFCPOLYLINE((#41,#42,#43,#44,#41));
#41=IFCCARTESIANPOINT((0.0E+00, 1.0E-01));
#42=IFCCARTESIANPOINT((2.8E+00, 1.0E-01));
#43=IFCCARTESIANPOINT((2.8E+00, -1.0E-01));
#44=IFCCARTESIANPOINT((0.0E+00, -1.0E-01));
```

9.1.3.3 Concept of shape aspect

In addition of having only an *IfcProductDefinitionShape* at *IfcProduct* (or any of its subtype) which is a kind of "black-box" container (i.e. a single semantic unit), the *IfcProductDefinitionShape* can be further semantically decomposed into shape aspects – each aspect is a part of the product representation which has a semantic meaning assigned.

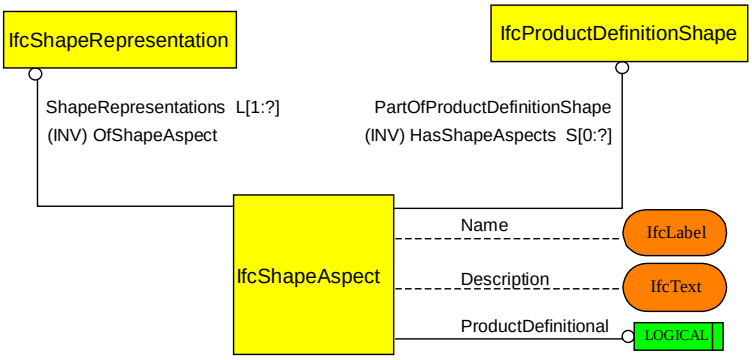


Figure 111 : Definition of shape aspect

An *IfcShapeAspect* is a concept for grouping of representation items that represent aspects (or components) of the shape of a product. That is, a shape aspect represents a distinctive part of a product that can be explicitly addressed as a shape representation. For instance, for an instance of *IfcDoor*, the shape representation of the door cill and the door swing might need to be addressed separately. Each of these can be defined as a shape aspect comprising distinct sets of geometric representation items.

Note: Still the whole shape representation of the IfcDoor is given as an IfcShapeRepresentation (within the Representations relationship at IfcProductDefinitionShape). Therefore applications are not necessarily required to understand this additional, more granular shape aspect structure in order to represent shape representations.

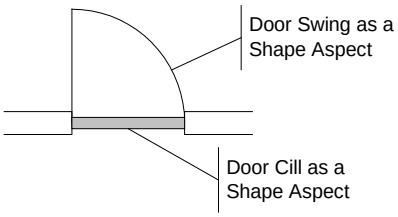


Figure 112 : Addressable shape aspects of a door

Note: The example files shows the three shape representations for the door, one for the whole, two for subsets (which does not have to be mutually exclusive nor have to sum up to the whole).

```
#1=IFCDOOR('abcdefghijklmnopqrst11', #2, $, $, $, #3, #100, $, $, $);

#10=IFCSHAPEASPECT((#200), 'Door Cill', $, .UNKNOWN., #100);
#11=IFCSHAPEASPECT((#201), 'Door Swing', $, .UNKNOWN., #100);

/* See elsewhere for guidance of product definition shape and shape representation */
#100=IFCPRODUCTDEFINITIONSHAPE ($, $, (#101));

/* See elsewhere for guidance of shape representation */
#101=IFCSHAPEREPRESENTATION (..); /* common shape representation of the door */
#200=IFCSHAPEREPRESENTATION (..); /* shape representation of the door cill   */
#201=IFCSHAPEREPRESENTATION (..); /* shape representation of the door swing  */
```

9.1.4 Concept of various shape representation types

Each of the *IfcRepresentation*, and particularly each of the *IfcShapeRepresentation*, are determined by the representation items, which are included within the *Items* attribute. There are three different ways to describe the items of an representation:

- by its (direct) geometric representation
- by its (direct) topological representation
- by its mapped representation, referring to a Cartesian transformation of a source representation

For the representation of shape, topological representation items are not used directly, but only through geometric representation items. Therefore only two UoF, the various forms of direct geometric representation and the mapped representation, are discussed here.

Example:

A B-rep of a shape uses the geometric representation item, such as the IfcFacetedBrep, which refers to one or several intermediate topological representation items, such as IfcClosedShell and IfcFace, before being geometrically established by IfcPolyLoop.

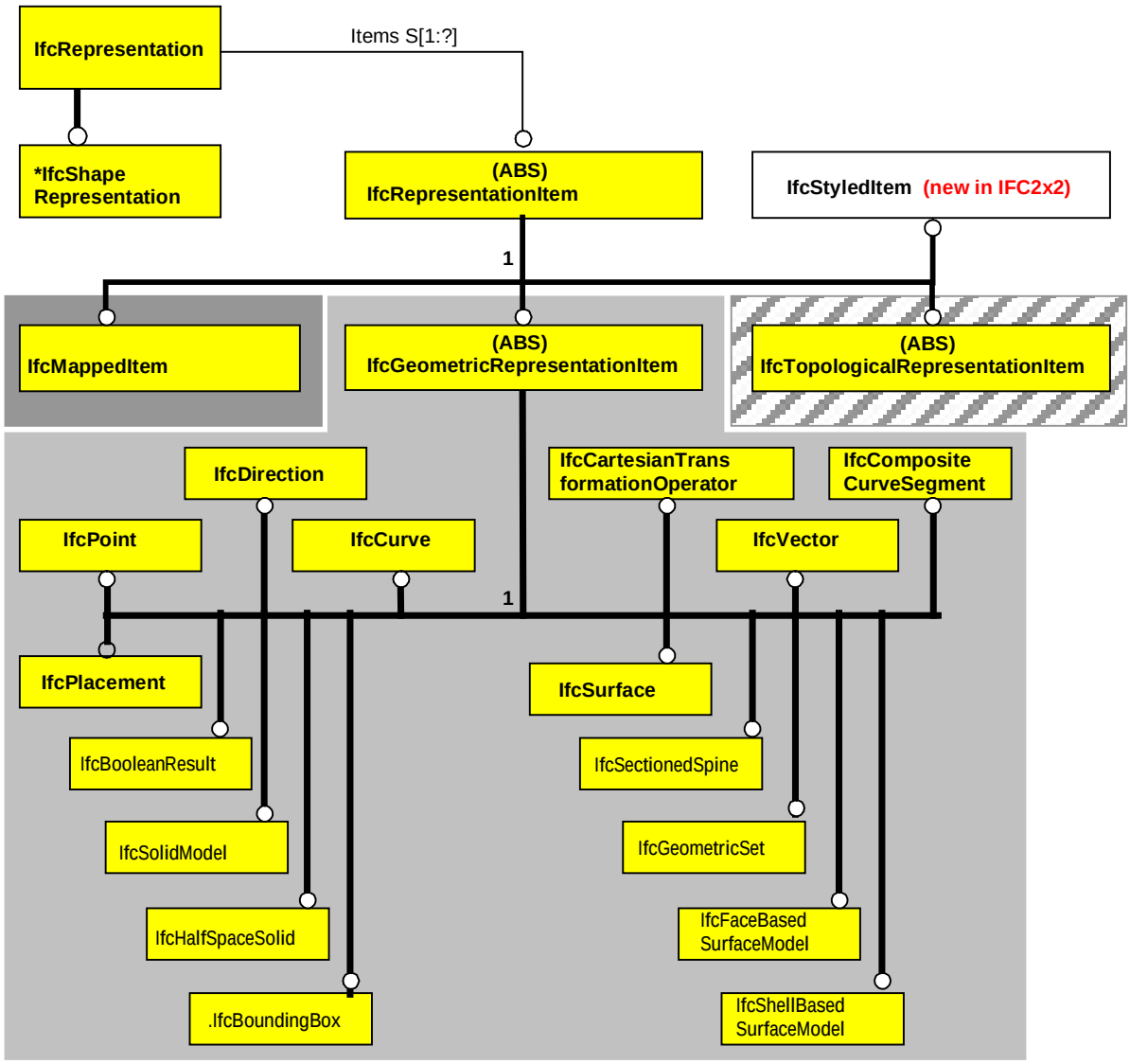


Figure 113 : Concept of shape representations

9.1.4.1 Concept of direct geometric representation

The attribute *Items* of *IfcRepresentation* (used also by *IfcShapeRepresentation*) allows to assign one or several instances of *IfcRepresentationItem* to describe the representation. For providing the concept of "Direct geometric representation", these instances are required to be all subtypes of *IfcGeometricRepresentationItem*. A further subdivision of the concept of "Direct geometric representation" is established by the various forms of the *RepresentationType* (given as attribute of *IfcRepresentation*).

The following representation types are distinguished:

- **Curve2D** – allows the use of 2D curves for the representation of line based representations, normally used to represent a path;
- **GeometricSet** – allows the use of points, curves, surfaces (both as 2D and 3D elements) for the shape representation (and it includes swept surfaces (both surfaces of extrusion and revolution) for the 3D geometric representation);
  - **GeometricCurveSet** – further restricts the representation type *GeometricSet* to only include points and curves (both as 2D and 3D elements) for the shape representation;
- **SurfaceModel** – allows the use of face based surface model and shell based surface model, which includes facetted face sets and shells (both open and closed shells) for the shape representation, surface models are always 3D representations;

- **SolidModel** – allows the use of solids, which includes swept solids, Boolean results and B-rep bodies for the shape representation, solid models are always 3D representations. More specific types are:
  - **Brep** – further restricts the representation type *SolidModel* to only include faceted B-rep with and without voids. B-rep models require the satisfaction of the stipulated Euler formulas. No other types of manifold boundary representations then faceted B-rep are currently in scope of IFC2x;
  - **SweptSolid** – further restricts the representation type *SolidModel* to only include swept area solids, including both swept solid by extrusion and swept solid by revolution;
  - **CSG** – further restricts the representation type *SolidModel* to only include Boolean results of operations between solid models, half spaces and other Boolean results. Operations include union, intersection and difference;
    - **Clipping** – further restricts the representation type *CSG* to only include Boolean results resulting in the clipping of an solid and thus restricting the Boolean operation to difference and the second operand to half space solid.
  - **AdvancedSweptSolid** – new in IFC2x2 - further restricts the representation type *SolidModel* to only include swept area solids created by sweeping a profile along a directrix – either a swept disk solid (sweeping of a circle along any 3D directrix) or a surface curve swept area solid (sweeping of any profile along a 3D directrix on a reference surface).
- **BoundingBox** – as an additional representation type it allows for a simplistic 3D representation given by a bounding box. It may be added as an additional representation to enable the display of the shape by application not being able to handle complex shape;
- **SectionedSpine** – as an additional representation type it allows for a cross section based representation of a spine curve and planar cross sections. It can represent a surface or a solid swept along any path (composed of straight lines and arcs) but the interpolations of the form between the cross sections is not defined. The *SectionedSpine* representation may be used within engineering applications for complex duct and pipe shapes (as an example).
- **MappedRepresentation** – a presentation based on a mapped item, referring to a representation map. It can be seen as an inserted block reference. The shape representation of the mapped item has a representation type declaring the type of its representation items.

9.1.4.1.1 Concept of Curve2D representation

The representation type *Curve2D* is used to provide a geometric representation of a path based on a bounded curve with a parametric range. It may only be used to provide 2D representation of a linear form (the path).

The *IfcBoundedCurve* should be used to provide the only item of *IfcShapeRepresentation.Items*. For multi-segmented path, the *IfcCompositeCurve* shall be used. The *Curve2D* representation is used, e.g. to provide the wall path information. A single line segment can be represented using an *IfcLine*, or (recommended) an *IfcPolyline*, referring to two *IfcCartesianPoint*'s.

See example based on Figure 109 within Section 9.1.3.2 for an example of how to use *Curve2D* representations.

9.1.4.1.2 Concept of Geometric Set representation

The representation type *GeometricSet* is used to provide a geometric representation based on a collection of points, lines and surfaces, where no topological structure is available. It may be used to provide either a 2D or a 3D representation of the form, where the items establishing the geometric set should be either all 2D or all 3D but not a mixture of both.

The *IfcGeometricSet* should be used to provide the only item of *IfcShapeRepresentation.Items*. An *IfcGeometricSet* contains one or several *Elements*, which are either subtypes of *IfcPoint*, *IfcCurve* or *IfcSurface*.

The representation type of *GeometricSet* is used, e.g., for the 2D representation of elements, such as furniture, equipment, distribution and building elements, or for special representations, such as the footprint representation of spaces. It also includes general 3D representations, where no solid model is given, and no topological information is available.

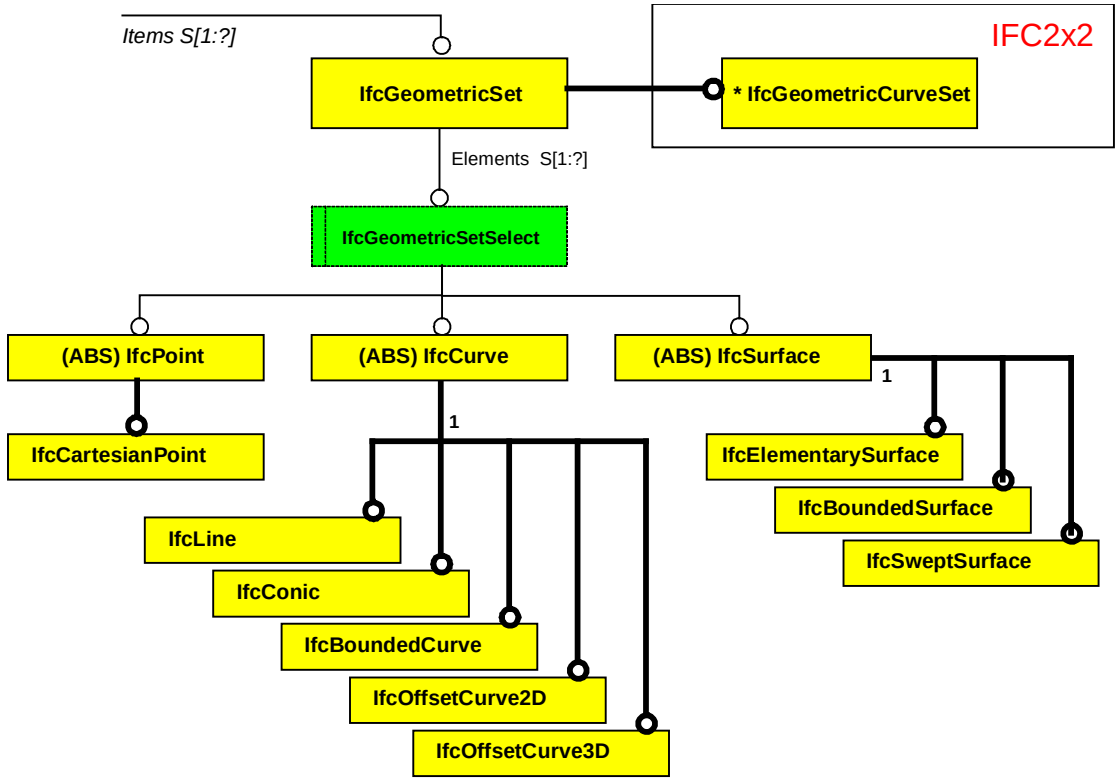


Figure 114 : Definition of Geometric Set representations

The use of geometric set representation for defining a 3D shape (without topological and solid information) includes the swept surface and the rectangular trimmed surface representation (mainly used for revolved surfaces with angle < 360').

Within IFC2x2 a new subtype of *IfcGeometricSet* has been introduced, the *IfcGeometricCurveSet*. It has an additional Where Rule, that restricts all *Elements* to be either points or curves, but not surfaces. This new subtype can be used to enforce a geometric set representation to be a point and curves only (e.g. a wire frame representation with no topological information).

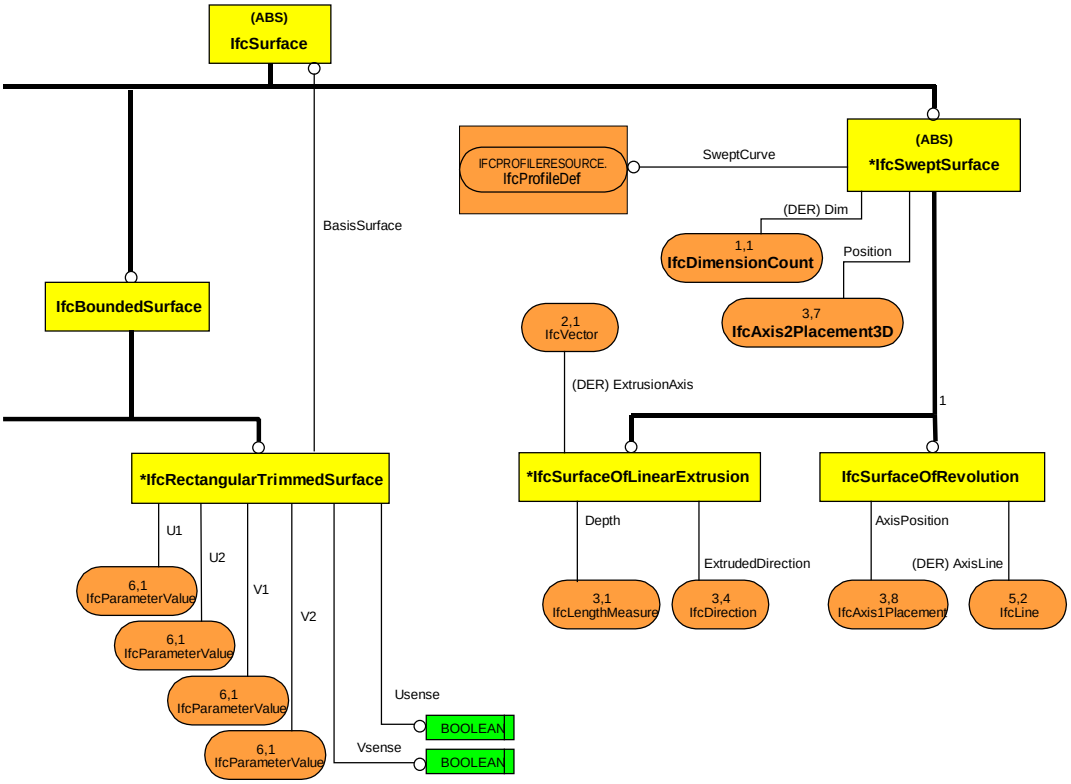


Figure 115 : Definition of surface representations within geometric set

Example:

The geometric representation of the "thin metal" circular duct (which should be given by a surface, not by a solid model) is given by extruding the profile, IfcCircleProfileDef, along a depth. The circle profile definition is a parametric profile, which is mapped into the XY plane of the position coordinate system of the swept surface and extruded (here perpendicular) to the XY plane by the depth.

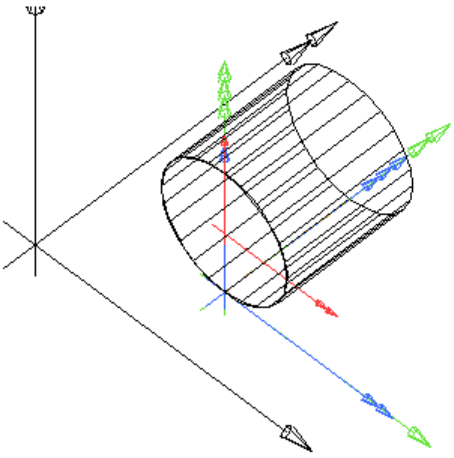


Figure 116 : Example of Geometric Set representation of a duct (using swept surface)

```
#1=IFCFLOWSEGMENT('abcdefghijklmnopqrst12', #2, $, $, $, #3, #4, $, .FLUIDFLOW.,
   .DUCTSEGMENT.);
#2=IFCOWNERHISTORY(#6, #7, .READWRITE., .NOCHANGE., $, $, $, 978921854);

/* UoF local absolute placement */
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.000000E+00, 1.000000E+00, 0.000000E+00));

/* UoF representation context */
```

```
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));

/* geometric set representation for the duct segment */
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11));
#11=IFCSHAPEREPRESENTATION(#12, '', 'GeometricSet', (#13));
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#16=IFCCARTESIANPOINT((2.000000E+00, 1.000000E+00, 0.000000E+00));

/* use of swept surface for the geometric set */
#13=IFCGEOMETRICSET((#17));
#17=IFCSURFACEOFLINEAREXTRUSION(#18, #21, #25, 2.000000E+00);
#18=IFCCIRCLEPROFILEDEF(.CURVE., $, #19, 1.000000E+00);
#19=IFCAXIS2PLACEMENT2D(#20, $);
#20=IFCCARTESIANPOINT((1.000000E+00, 0.000000E+00));
#21=IFCAXIS2PLACEMENT3D(#22, #23, #24);
#22=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#23=IFCDIRECTION((0.000000E+00, 1.000000E+00, 0.000000E+00));
#24=IFCDIRECTION((0.000000E+00, 0.000000E+00, 1.000000E+00));
#25=IFCDIRECTION((0.000000E+00, 0.000000E+00, 1.000000E+00));
```

9.1.4.1.2.1 Concept of Geometric Curve Set representation

The representation type *GeometricCurveSet* is used to provide a geometric representation based on a collection of points and lines, where no topological structure is available. It may be used to provide either a 2D or a 3D representation of the form, where the items establishing the geometric set should be either all 2D or all 3D but not a mixture of both.

The representation type *GeometrcCurveSet* therefore further restricts the type *GeometricSet*, its use is foreseen in future, if further distinctions of geometric sets are required by implementations. Currently the use of geometric curve set is proposed for all cases of point and line representations, that provides 2D representations of elements.

Since IFC2x2 the new subtype *IfcGeometricCurveSet* shall be used to express a collection of points and curves only.

Example:

> *In many cases simple 2D representations of furniture and equipment are sufficient, or may be given in addition to the 3D surface or solid model representations. In this example, the shape of a furniture of type 'Chair' is represented as a 2D geometric set, established within the XY plane of the object coordinate system. The geometric set contains a set of line elements, provided as IfcPolyline's.*

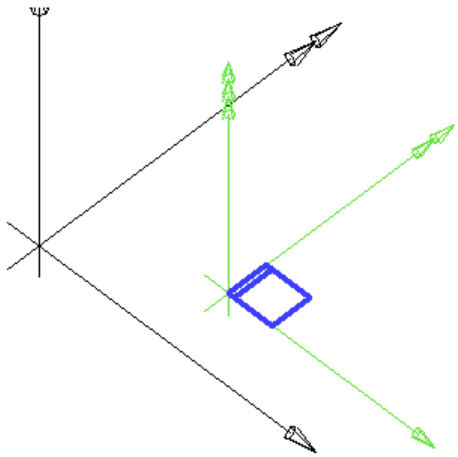


Figure 117 : Example of Geometric Curve Set representation of furniture

```
#1=IFCFURNITUREELEMENT('abcdefghijklmnopqrst02', #2, $, $, $, #3, #4, $);
#2=IFCOWNERHISTORY(#6, #7, .READWRITE., .NOCHANGE., $, $, $, 978921854);

/* UoF local absolute placement */
#3=IFCLOCALPLACEMENT($, #10);
```

```
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.000000E+00, 1.000000E+00, 0.000000E+00));

/* UoF representation context */
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));

/* geometric set representation for the furniture */
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11));
#11=IFCSHAPEREPRESENTATION(#12, '', 'GeometricCurveSet', (#18));
#18=IFCGEOMETRICCURVESET((#19, #24));
/* #18=IFCGEOMETRICSET((#19, #24)); -- until IFC2x */
#19=IFCPOLYLINE((#20, #21, #22, #23, #20));
#20=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#21=IFCCARTESIANPOINT((0.700000E+00, 0.000000E+00, 0.000000E+00));
#22=IFCCARTESIANPOINT((0.700000E+00, 0.600000E+00, 0.000000E+00));
#23=IFCCARTESIANPOINT((0.000000E+00, 0.600000E+00, 0.000000E+00));
#24=IFCPOLYLINE((#25, #26));
#25=IFCCARTESIANPOINT((0.100000E+00, 0.000000E+00, 0.000000E+00));
#26=IFCCARTESIANPOINT((0.100000E+00, 0.600000E+00, 0.000000E+00));
```

9.1.4.1.3 Concept of Surface Model representation

The representation type *SurfaceModel* is used to provide a geometric representation based on a collection of faces, provided by face bounds based on poly loops, where a topological structure is available. It may be used to provide a 3D representation of the form. It is normally used to describe the (explicit) 3D form of an object, where no volume information is available or desired.

> *NOTE: Often the rigid solid B-rep representation is not applicable, e.g. for "thin metal" structures, as cupper plates, etc. where an open shell is more applicable to describe the form, or the B-rep representation can not be satisfied by applications without a sophisticated geometry engine. In these cases the SurfaceModel may provide an alternative to exchange the 3D explicit shape.*

The representation type *SurfaceModel* is used, e.g., for the shape representation of distribution (flow) elements, metal plates and other elements.

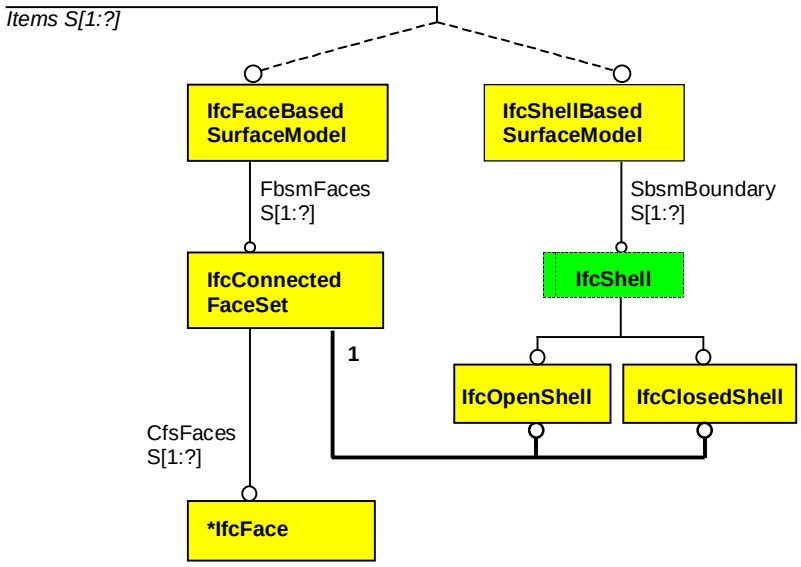


Figure 118 : Definition of Surface Model representations

> *NOTE: Topological representation items, such as IfcOpenShell, IfcClosedShell, and IfcConnectedFaceSet are not allowed to be immediately referenced by the Items attribute of IfcShapeRepresentation. It is required, that an intermediate geometric representation item, either IfcFaceBasedSurfaceModel or IfcShellBasedSurfaceModel, has to be used.*

Example:

> *Often the use of surface models are more appropriate for building service equipment, although it would not allow clash detection based on the volume of elements. Surface models my also be*

used for terrain models. The example shows a simple surface model of a duct, given by the IfcDistributionFlowSegment.

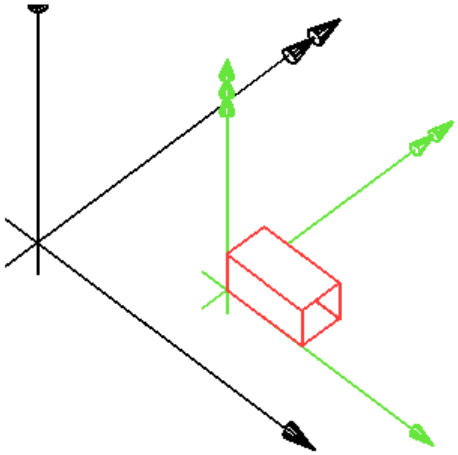


Figure 119 : Example of Surface Model representation of a piece of duct

NOTE: Within the example, a face based surface model had been used, where also a shell based surface model (based on an open shell) would have been available. The decision was arbitrary for the example.

```
#1=IFCFLOWSEGMENT('abcdefghijklmnopqrst02', #2, $, $, $, #3, #4, $);
#2=IFCOWNERHISTORY(#6, #7, .READWRITE., .NOCHANGE., $, $, $, 978921854);

/* UoF local absolute placement */
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.000000E+00, 1.000000E+00, 0.000000E+00));

/* UoF representation context */
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));

/* surface model representation for the duct */
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11));
#11=IFCSHAPEREPRESENTATION(#12, '', 'SurfaceModel', (#18));
#18=IFCFACEBASEDSURFACEMODEL((#19));
#19=IFCCONNECTEDFACESET((#23, #21, #22, #20));

#20=IFCFACE((#24));
#21=IFCFACE((#25));
#22=IFCFACE((#26));
#23=IFCFACE((#27));
#24=IFCFACEOUTERBOUND(#28, .T.);
#25=IFCFACEOUTERBOUND(#29, .T.);
#26=IFCFACEOUTERBOUND(#30, .T.);
#27=IFCFACEOUTERBOUND(#31, .T.);
#28=IFCPOLYLOOP((#40, #41, #42, #43));
#29=IFCPOLYLOOP((#50, #51, #52, #53));
#30=IFCPOLYLOOP((#60, #61, #62, #63));
#31=IFCPOLYLOOP((#70, #71, #72, #73));
#40=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#41=IFCCARTESIANPOINT((1.200000E+00, 0.000000E+00, 0.000000E+00));
#42=IFCCARTESIANPOINT((1.200000E+00, 0.000000E+00, 6.000000E-01));
#43=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 6.000000E-01));
#50=IFCCARTESIANPOINT((0.000000E+00, 6.000000E-01, 0.000000E+00));
#51=IFCCARTESIANPOINT((0.000000E+00, 6.000000E-01, 6.000000E-01));
#52=IFCCARTESIANPOINT((1.200000E+00, 6.000000E-01, 6.000000E-01));
#53=IFCCARTESIANPOINT((0.000000E+00, 6.000000E-01, 0.000000E+00));
#60=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 6.000000E-01));
#61=IFCCARTESIANPOINT((1.200000E+00, 0.000000E+00, 6.000000E-01));
#62=IFCCARTESIANPOINT((1.200000E+00, 6.000000E-01, 6.000000E-01));
#63=IFCCARTESIANPOINT((0.000000E+00, 6.000000E-01, 6.000000E-01));
#70=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#71=IFCCARTESIANPOINT((0.000000E+00, 6.000000E-01, 0.000000E+00));
#72=IFCCARTESIANPOINT((1.200000E+00, 6.000000E-01, 0.000000E+00));
#73=IFCCARTESIANPOINT((1.200000E+00, 6.000000E-01, 6.000000E-01));
```

9.1.4.1.4 Concept of Solid Model representation

A solid model is the most complete representation of the nominal shape of a product. It determines that all points in the interior are connected and that any point in the domain can be classified as being inside, outside or on the boundary of a solid.

Within the IFC2x model there are three different types of solid model representations defined:

- Boundary model representation (restricted to faceted, manifold b-rep with or without voids);
- Swept area solid representation (using either extrusion or revolution);
- Constructive solid geometry representation, represented by their component solids and the sequence of Boolean operations (it maybe further restricted to solely clipping operations).

The three different specializations of the *SolidModel* representation type then require the correct usage of the different subtypes of *IfcSolidModel* to provide the shape representation of the product.

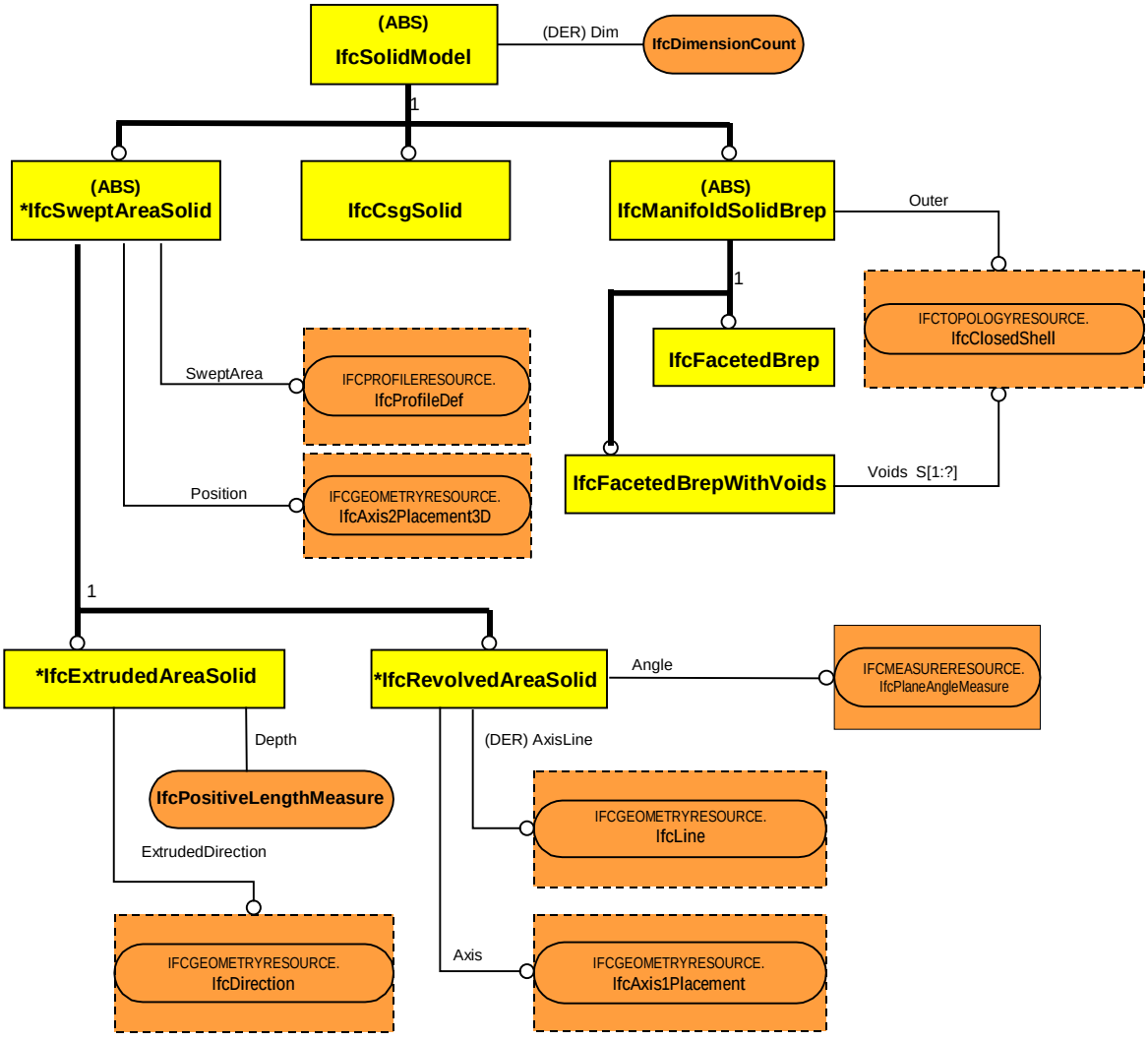


Figure 120 : Definition of IfcSolidModel with subtypes (without CSG)

The *IfcSweptAreaSolid*, when referencing parameterized profile definitions, is normally used as the preferred geometric representation for building elements.

9.1.4.1.4.1 Concept of B-rep representation

The representation type 'Brep' is used to provide a geometric representation based on a collection of faces, provided by face bounds based on poly loops, where a topological structure is available, and

the interior (or volume) is defined. Therefore the topological normal of the B-rep at each point of its boundary is the surface normal direction that points away from the solid material. Therefore all faces needs to be (explicitly or implicitly) oriented. All Euler formulas need to the satisfied for the B-rep.

It may be used to provide a 3D representation of the form. It is normally used to describe the (explicit) 3D form of an object, where volume information is available or desired. The exchange of B-rep shape allows for the exchange of (almost) every shape for elements in building & construction.

B-rep representations are restricted to faceted boundary representations only within the IFC2x model. Therefore the only form for face bounds are polyloops, curvatures have to be faceted using approximation techniques.

B-rep representation requires the complete definition of the boundaries of the shape. E.g. all edges of the B-rep are shared by (at least) two faces in the closed shell. In order to determine that the shell is closed and the same edge is referenced twice, the coordinates of the vertices have to be coincident. This can either be achieved by sharing the same instance of the *IfcCartesianPoint* by two or more face bounds (provided through the *IfcPolyLoop* entity), or by using identical copies of the *IfcCartesianPoint*. In the latter case, the *Precision* attribute of the *IfcGeometricRepresentationContext* determines, which is the maximum difference, under which two points are still be considered to be identical.

Example:

The standard geometric representation of any (non-standard) building element, provided by the IfcBuildingElementProxy, is the Brep representation. The following example shows the Brep shape representation of a proxy with the shape of a box.

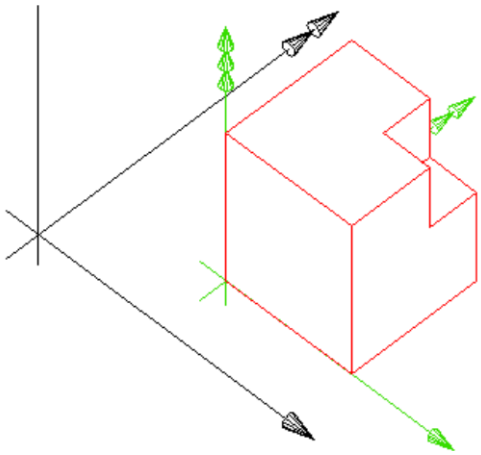


Figure 121 : Example of B-rep Model representation of a proxy

NOTE: The example shows the reference of polyloop vertices by multiple instance reference.[16]

```
#1=IFCBUILDINGELEMENTPROXY('abcdefghijklmnopqrst02', #2, 'Box', $, $, #3, #4, $,$);
#2=IFCOWNERHISTORY(#6, #7, .READWRITE., .NOCHANGE., $, $, $, 978921854);

/* UoF local absolute placement */
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.0, 1.0, 0.0));

/* UoF representation context */
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, 1.0E-06, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.0, 0.0, 0.0));

/* b-rep model representation for the duct */
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11));
#11=IFCSHAPEREPRESENTATION(#12, '', 'Brep', (#174));
#174=IFCFACETEDBREP(#173);
#173=IFCCLOSEDSHELL((#148,#151,#154,#157,#160,#163,#166,#169,#172));
```

16 In contrary to the previous examples, this example uses "normal" REAL value formats, instead of the notation with exponents in other examples. Both are valid instantiations of the REAL data type and may be used.

```
#132=IFCCARTESIANPOINT((0.25,0.25,1.5));
#133=IFCCARTESIANPOINT((1.,0.25,1.5));
#134=IFCCARTESIANPOINT((1.,1.,1.5));
#135=IFCCARTESIANPOINT((0.25,1.,1.5));
#136=IFCCARTESIANPOINT((0.25,0.25,2.5));
#137=IFCCARTESIANPOINT((1.,0.25,2.5));
#138=IFCCARTESIANPOINT((0.25,1.,2.5));
#139=IFCCARTESIANPOINT((-1.,-1.,0.));
#140=IFCCARTESIANPOINT((1.,-1.,0.));
#141=IFCCARTESIANPOINT((1.,1.,0.));
#142=IFCCARTESIANPOINT((-1.,1.,0.));
#143=IFCCARTESIANPOINT((-1.,-1.,2.5));
#144=IFCCARTESIANPOINT((1.,-1.,2.5));
#145=IFCCARTESIANPOINT((-1.,1.,2.5));
#146=IFCPOLYLOOP((#138,#136,#132,#135));
#147=IFCFACEOUTERBOUND(#146,.T.);
#148=IFCFACE((#147));
#149=IFCPOLYLOOP((#136,#137,#133,#132));
#150=IFCFACEOUTERBOUND(#149,.T.);
#151=IFCFACE((#150));
#152=IFCPOLYLOOP((#132,#133,#134,#135));
#153=IFCFACEOUTERBOUND(#152,.T.);
#154=IFCFACE((#153));
#155=IFCPOLYLOOP((#137,#136,#138,#145,#143,#144));
#156=IFCFACEOUTERBOUND(#155,.T.);
#157=IFCFACE((#156));
#158=IFCPOLYLOOP((#138,#135,#134,#141,#142,#145));
#159=IFCFACEOUTERBOUND(#158,.T.);
#160=IFCFACE((#159));
#161=IFCPOLYLOOP((#133,#137,#144,#140,#141,#134));
#162=IFCFACEOUTERBOUND(#161,.T.);
#163=IFCFACE((#162));
#164=IFCPOLYLOOP((#143,#145,#142,#139));
#165=IFCFACEOUTERBOUND(#164,.T.);
#166=IFCFACE((#165));
#167=IFCPOLYLOOP((#144,#143,#139,#140));
#168=IFCFACEOUTERBOUND(#167,.T.);
#169=IFCFACE((#168));
#170=IFCPOLYLOOP((#140,#139,#142,#141));
#171=IFCFACEOUTERBOUND(#170,.T.);
#172=IFCFACE((#171));
```

9.1.4.1.4.2 Concept of Swept Solid representation

The representation type *SweptSolid* is used to provide a geometric representation based on sweeping a profile (given by a planar bounded area). There are two different types of sweeping operations:

- linear extrusion
- revolution

The position of the swept body depends on the axis placement of the swept area solid, where the XY plane of the placement is used to place the profile.

9.1.4.1.4.3 Concept of CSG representation

The representation type *CSG* is used to provide a geometric representation based on the CSG model of the represented object. A solid represented as a CSG model is defined by a collection of so-called primitive solids, combined using regularized Boolean operations.

The use of CSG is currently restricted to the Boolean operations on other solid models, as no CSG primitives are included in the IFC2x specification. The provision of CSG representations mainly allows for clipping operations between swept area solids and half space solids.

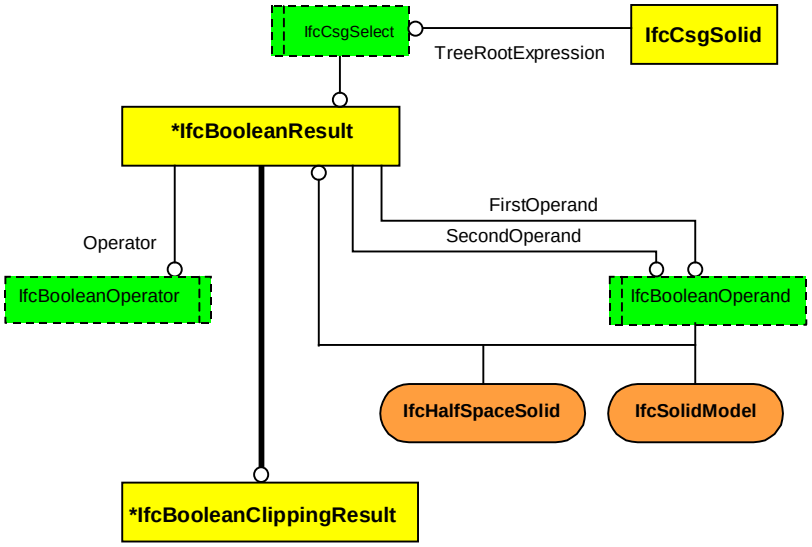


Figure 122 : Definition of Boolean results

NOTE: Current Geometry use definitions do not foresee the use of other Boolean operations, than the clipping (expressed by a Boolean difference), therefore no Boolean unions or intersections are momentarily used.

9.1.4.1.4.3.1 Concept of Clipping representation

The representation type *Clipping* is used to provide a geometric representation based on the CSG model of the represented object, where the first operand is a solid model, or another Boolean result within the CSG tree and the second operand is a half space solid. The Boolean operator is always DIFFERENCE.

The representation type *Clipping* is normally used to clip part of the solid (given by a swept area solid), whereas several clippings allowed within the CSG tree. The only representation item within the list of *Items* is the *IfcBooleanClippingResult*.

```
ENTITY IfcBooleanClippingResult
   SUBTYPE OF (IfcBooleanResult);
   WHERE
      WR1: ('IFCGEOMETRICMODELRESOURCE.IFCSWEPTAREASOLID' IN TYPEOF(FirstOperand)) OR
        ('IFCGEOMETRICMODELRESOURCE.IFCBOOLEANCLIPPINGRESULT' IN TYPEOF(FirstOperand));
      WR2: ('IFCGEOMETRICMODELRESOURCE.IFCHALFSPACESOLID' IN TYPEOF(SecondOperand));
      WR3: Operator = DIFFERENCE;
END_ENTITY;
```

The where rules restrict the clipping to proper Boolean operands and operator. The following example shows the use of clipping for a wall body. The wall body is represented by a swept solid (as explained in 9.1.4.1.4.2) and the clipping by an *IfcHalfSpaceSolid*. There are two different types of clipping solids, the unlimited *IfcHalfSpaceSolid* (or the computationally optimized *IfcBoxedHalfSpace*) and the limited *IfcPolygonalBoundedHalfSpace*.

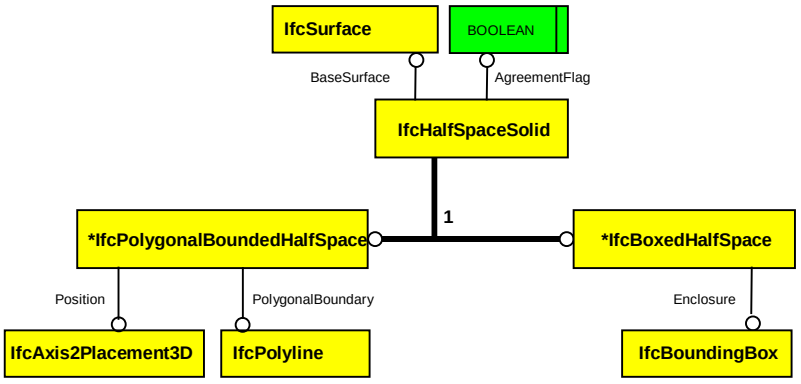


Figure 123 : Definition of half space solids

Example shows a clipped wall, based on the example given in Figure 109.

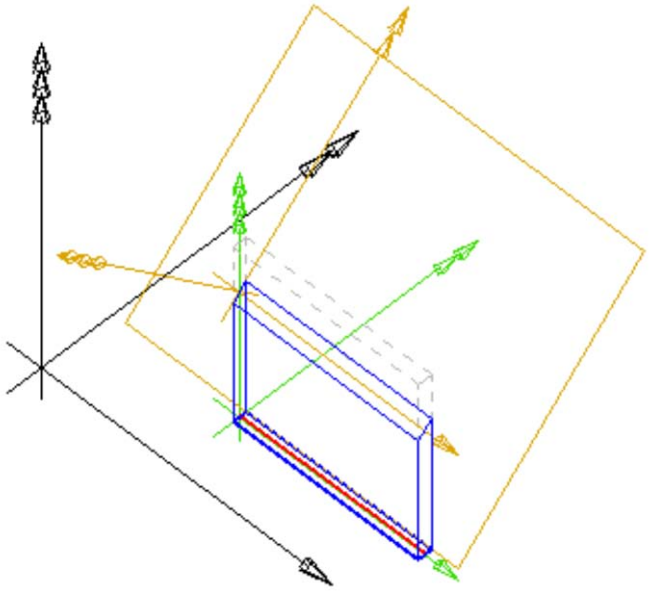


Figure 124 : Use of clipping for a wall body

The IFC file would look like:

```
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);

#3=IFCLOCALPLACEMENT($, #10);
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11, #13));
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.0E+00, 1.0E+00, 0.0E+00));
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));

/* shape representation of the wall axis */
#11=IFCSHAPEREPRESENTATION(#12, 'Axis', 'Curve2D', (#18));
#18=IFCTRIMMEDCURVE(#19,(#20),(#21),.T.,.CARTESIAN.);
#19=IFCLINE(#30, #31);
#30=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#31=IFCVECTOR(#32,2.8E+00);
#32=IFCDIRECTION((1.0E+00, 0.0E+00));
#20=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#21=IFCCARTESIANPOINT((2.80E+00, 0.0E+00));

/* shape representation of the clipped body */
#13=IFCSHAPEREPRESENTATION(#12, 'Body', 'Clipping', (#50));
#50=IFCBOOLEANCLIPPINGRESULT(.DIFFERENCE., #22, #51);

/* geometric representation of the extruded solid */
#22=IFCEXTRUDEDAREASOLID(#23, #26, #29, 2.80E+00);
#26=IFCAXIS2PLACEMENT3D(#28, $, $);
#28=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));
#29=IFCDIRECTION((0.0E+00, 0.0E+00, 1.0E+00));
#23=IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #40);
#40=IFCPOLYLINE((#41,#42,#43,#44,#41));
#41=IFCCARTESIANPOINT((0.0E+00, 1.0E-01));
#42=IFCCARTESIANPOINT((2.8E+00, 1.0E-01));
#43=IFCCARTESIANPOINT((2.8E+00, -1.0E-01));
#44=IFCCARTESIANPOINT((0.0E+00, -1.0E-01));

/* geometric representation of the clipping plane */
#51=IFCHALFSPACESOLID(#52, .F.);
#52=IFCPLANE(#53);
#53=IFCAXIS2PLACEMENT3D(#54, #55, #56);
#54=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 2.0E+00));
#55=IFCDIRECTION((0.0E+00, -0.7070106E+00, 0.7070106E+00));
#56=IFCDIRECTION((1.0E+00, 0.0E+00, 0.0E+00));
```

9.1.4.1.5 Concept of Bounding Box representation

9.1.4.1.6 Concept of Sectioned Spine representation

9.1.4.2 Mapped representations

The IFC2x specification allows for the use of mapped representations in order to reuse the shape of a particular object type at all particular instances of this object. The concept of mapped representations can be compared with the concept of a block[17] definition and block insertions within most of CAD systems. The block contains geometric items within a local (block) coordinate system, which can be inserted one or several times into the actual drawing model by using a block reference, including a Cartesian transformation operator, which normally includes scaling.

A similar concept is used in IFC2x, where *IfcRepresentationMap* represents the block definition, and *IfcMappedItem* represents the block insertion. The *IfcRepresentationMap* can include any representation by containing one or several *IfcGeometricRepresentationItem*.

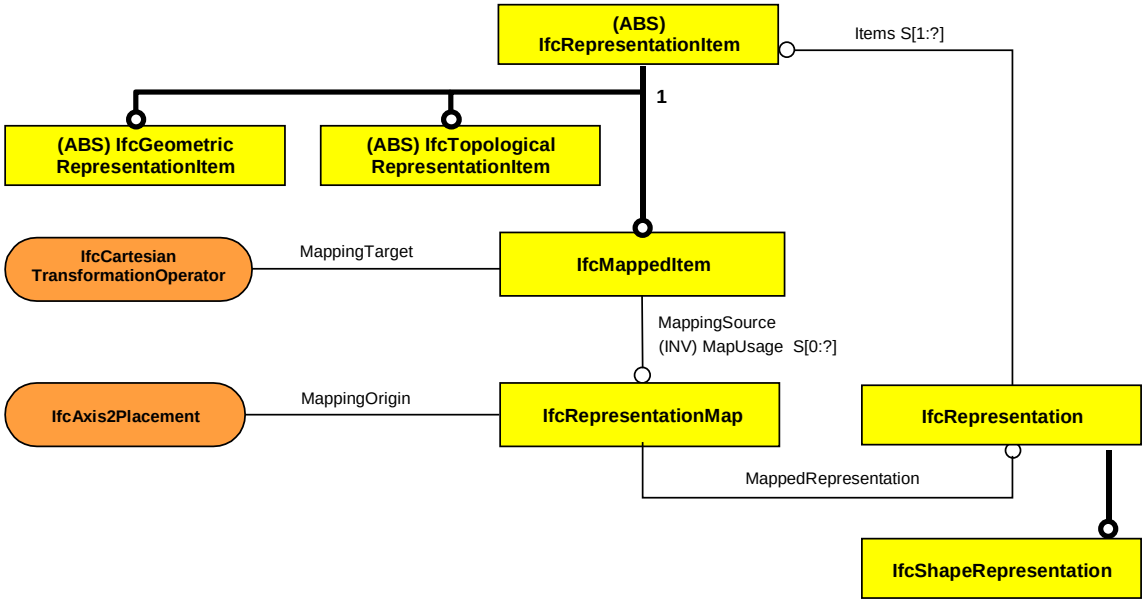


Figure 125 : Definition of mapped items and representation maps

> *NOTE: Similar as in CAD systems, the extensive use of representation maps helps to reduce the file size of the IFC exchange and could help to improve the performance of the received file, if the reading application does convert the representation maps into its internal block/cell/macro structure.*

The *MappedRepresentation* is usually of type *IfcShapeRepresentation* (although in principle it could also be of type *IfcMappedItem*, and thereby it provides for the capability of nested blocks – but this is not further elaborated here). Any *RepresentationType* (as discussed in 9.1.4.1) can be used as a *MappedRepresentation*.

Example:

> *A sample use of representation maps shows the reuse of geometry for a (very simplified) flow terminal, like an air outlet. It reuses the geometry as shown in Figure 118.*
>
> *Instead of assigning the same geometry item instances to all instances of the various air outlets, all reference it by sharing the same representation map. The example includes mapped representations using uniform scaling and non-uniform scaling of the original representation map.*

[17] Beside the term "block", other terms, like cell, macro or library part are used to denote the same or similar concepts

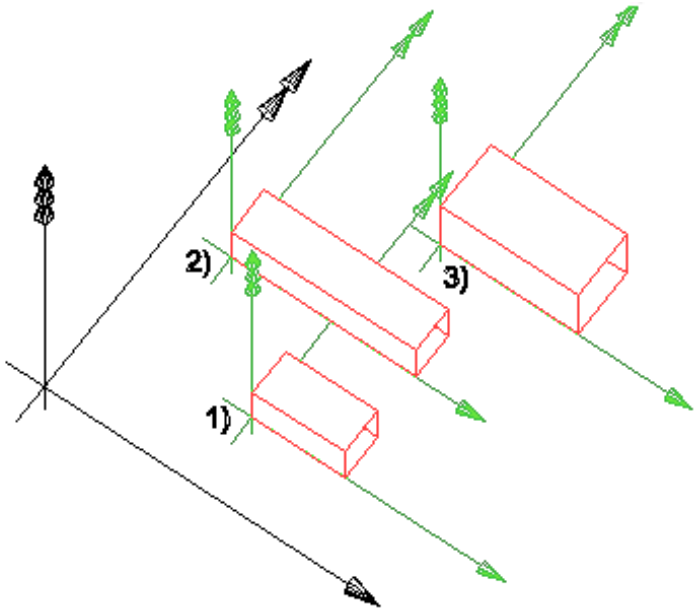


Figure 126 : Example for mapped items

The three instances of IfcFlowSegment uses different IfcMappedItem.MappingTarget. Segment [1] has an uniform scale of 1.0, segment [2] a non-uniform scale of 2.0,1.0,1.0 and segment [3] an uniform scale of 1.5.

```
/* placement and block insertion for the first flow segment */
#10=IFCFLOWSEGMENT('abcdefghijklmnopqrst02', #1000, $, $, $, #11, #14, $);
#11=IFCLOCALPLACEMENT($, #12);
#12=IFCAXIS2PLACEMENT3D(#13, $, $);
#13=IFCCARTESIANPOINT((2.0E+00, 1.0E+00, 0.0E+00));
#14=IFCPRODUCTDEFINITIONSHAPE($, $, (#15));
#15=IFCSHAPEREPRESENTATION(#100, '', 'MappedRepresentation', (#16));
#16=IFCMAPPEDITEM(#105,#17);
#17=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#18,$,$);
#18=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));

/* placement and block insertion for the second flow segment */
#20=IFCFLOWSEGMENT('abcdefghijklmnopqrst03', #1000, $, $, $, #21, #24, $);
#21=IFCLOCALPLACEMENT($, #22);
#22=IFCAXIS2PLACEMENT3D(#23, $, $);
#23=IFCCARTESIANPOINT((1.0E+00, 2.0E+00, 1.0E+00));
#24=IFCPRODUCTDEFINITIONSHAPE($, $, (#25));
#25=IFCSHAPEREPRESENTATION(#100, '', 'MappedRepresentation', (#26));
#26=IFCMAPPEDITEM(#105,#27);
#27=IFCCARTESIANTRANSFORMATIONOPERATOR3DNONUNIFORM($,$,#28,2.0,$,1.0,1.0);
#28=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));

/* placement and block insertion for the third flow segment */
#30=IFCFLOWSEGMENT('abcdefghijklmnopqrst03', #1000, $, $, $, #31, #34, $);
#31=IFCLOCALPLACEMENT($, #32);
#32=IFCAXIS2PLACEMENT3D(#33, $, $);
#33=IFCCARTESIANPOINT((2.0E+00, 1.0E+00, 0.0E+00));
#34=IFCPRODUCTDEFINITIONSHAPE($, $, (#35));
#35=IFCSHAPEREPRESENTATION(#100, '', 'MappedRepresentation', (#36));
#36=IFCMAPPEDITEM(#105,#37);
#37=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#38,1.5,$);
#38=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));

/* shared definition of the block */
#105=IFCREPRESENTATIONMAP(#106,#111);
#106=IFCAXIS2PLACEMENT3D(#107, $, $);
#107=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));
#111=IFCSHAPEREPRESENTATION(#100, '', 'SurfaceModel', (#118));
#118=IFCFACEBASEDSURFACEMODEL((#119));
#119=IFCCONNECTEDFACESET((#123, #121, #122, #120));
/* further geometry definition skipped */
```

9.2 Concept of 2D representations and presentations

The new IFC2x2 provides for the capability to capture and exchange also 2D representations, not only geometric items (which had been available before) but also annotation items, such as text, hatching, tiling, dimensioning. In addition it offers a range of presentation capabilities to describe the system (or appearance) of symbols, curves, surfaces, filled areas and text.

The IFC2x2 model therefore allows to include all geometric and graphical content which is part of the model space[18], i.e. given within a single geometric representation context and not subjected to any projection. The paper space, i.e. the projection by a camera model onto one or several 2D sheets, and the additional annotations of such a paper space are both out of scope for IFC2x2.

2D representations and presentations can be used for:

- representation and presentation of semantic objects (or elements), such as building elements, building service elements, electrical elements, equipment elements, etc.
- representation and presentation of “free” geometry and annotations that are not attached to any semantic object

The 2D representations and presentations can be given as a default 2D representation to each semantic object or annotation, or can be defined scale and projection dependent. The same geometric representation can have multiple presentations, different for selected scale and projection specific geometric representation contexts. This provides for the capability to handle “multi-view blocks” or “view dependent macros, or cells” which are now part of modern CAD systems.

Note: In order to offer backward compatibility with existing IFC2x applications, complying to the IFC2x coordination view, the provision of presentation information for semantic objects shall not interfere with the established concept of shape representation.

The usage of all new 2D drafting capabilities, that have been added within the IFC2x2 model, has to follow the same principles, as the general geometric representations, introduced in section 2.

- it has to be included into the IFC product definition structure, including the *ObjectPlacement* and *Representation* concepts of *IfcProduct*. There is not geometric representation outside of an *IfcProduct* definition,
- it has to be encapsulated (as any geometry is encapsulated in the *IfcProduct* definition),
  - the (potentially multiple) presentations of semantic objects are handled as styled representations included in the (subtype of) *IfcProduct.Representation*, in addition to the “master” geometric representation, as shown in 9.1.3.
  - the (potentially multiple) presentations of 2D representation of “free” annotations, i.e. those geometries that are independent of any product representation (such as free text, additional lines, non-associated dimensions, etc) are representations of an new *IfcAnnotation* object. It is used to encapsulate those “unassigned” graphical presentations.

The next section describes the principle strategy to include the presentation capabilities in a backward compatible fashion in order to guarantee the platform stability of the IFC2x platform. The Figure 126 shows the new entities introduced in IFC2x2 by using the brown color filling.

[18] The model space is used here to differentiate it from the 2D projection, often referred to as paper space or plots. The model space is characterized that it does not have an original scale and refers to a single world coordinate system.

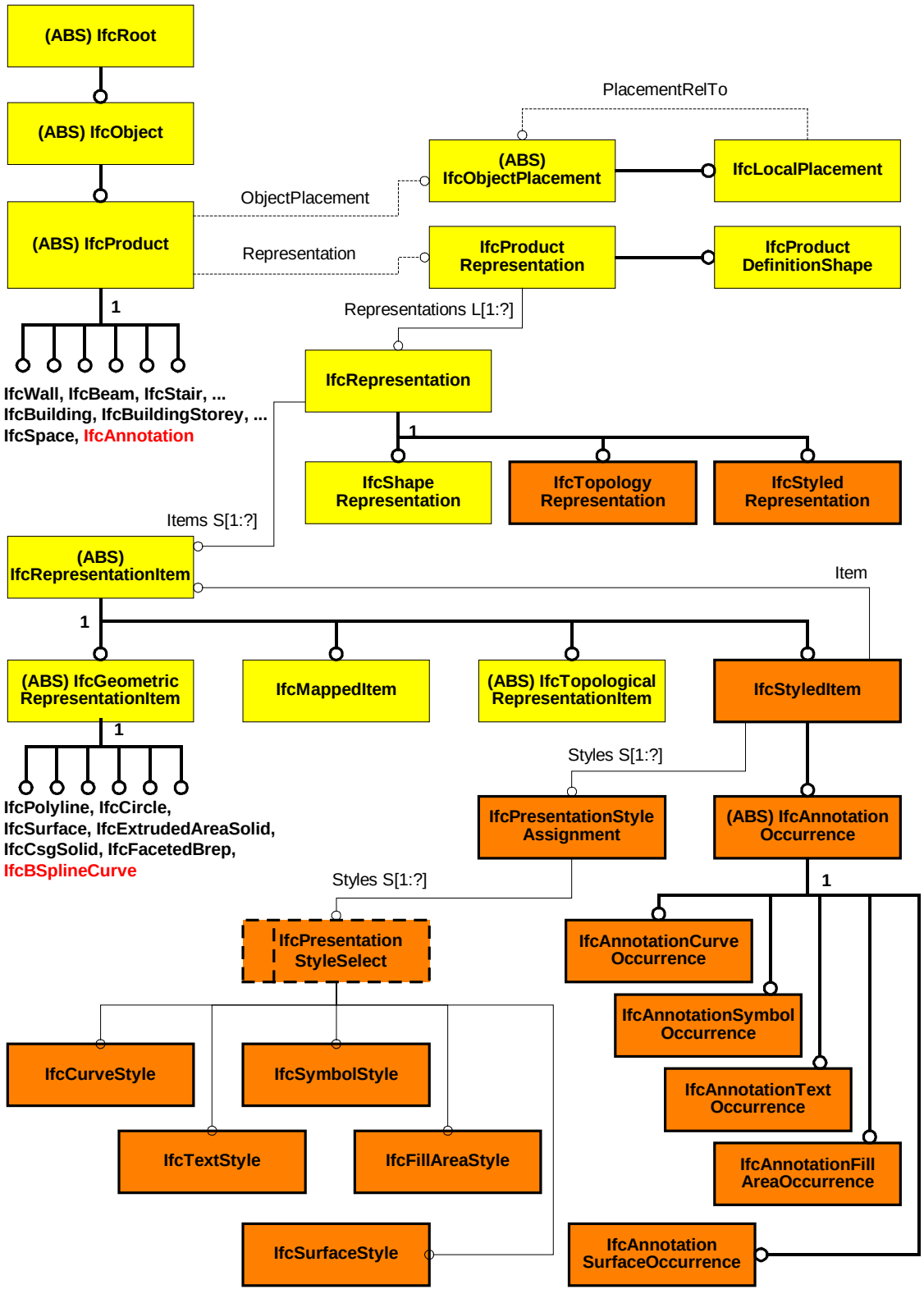


Figure 127 : entities for the presentation model of IFC2x2

9.2.1 Presentation of semantic objects

Each representation is included (or encapsulated – following the object-oriented principles) within the definition of an individual semantic object – being either a product occurrence, i.e. subtypes of *IfcProduct*), or a product type (or block) then a subtype of *IfcTypeProduct*.

Each geometric representation (*IfcShapeRepresentation*) is defined in its own object coordinate system, in the case of product occurrences, the object coordinate system is placed through a local placement (*IfcObjectPlacement*) either directly into the world coordinate system, or through some intermediate object placements. Each semantic object can have zero, one or many geometric representations, each being contained in a separate instance of *IfcShapeRepresentation*, but all are placed by a single instance of *IfcObjectPlacement*.

In order to maintain backward compatibility, each subtype of *IfcProduct* (with the exception of *IfcAnnotation*) should have at least one *IfcShapeRepresentation* within the List of *IfcProductDefinitionShape.Representations*. In addition to that, it may have zero, one or many *IfcStyledRepresentation*, each including one or more subtypes of *IfcStyledItem*.

The *IfcStyledItem* is the container for a styled representation (i.e. a representation with presentation information). It may contain *IfcGeometricRepresentationItem*'s within its list of *Items*, or may reference *IfcGeometricRepresentationItem*'s from another shape representation of the same product. The following gives a simple example for a wall, having shape representations and a presentation information for the axis (given separately to maintain backward compatibility).

Note: The surface color is only partially shown, but the example uses a unique color (R:247, G:198, B:0) for the whole body surface and a unique edge color (R:179 , G:179, B:179) for the wall.

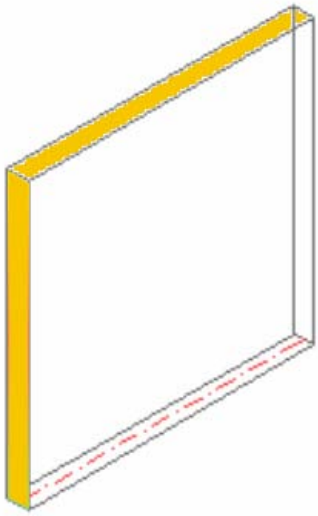


Figure 128 : Representation of the wall example

```
/* wall object with local placement and relationships */
#53=IFCWALLSTANDARDCASE('0vZnM3yZjD48ZdN7dQ17oA',#16,$,'IfcWall #1',$,#28,#37,$);
#59=IFCRELCONTAINEDINSPATIALSTRUCTURE('2$WrbX8KbDhRdQlWgOqya4',#16,$,$,(#53),#23);
#58=IFCRELASSOCIATESMATERIAL('257hs0LEjCXB2lXfXjwIIL',#16,$,$,(#53),#57);
#37=IFCPRODUCTDEFINITIONSHAPE($,$,(#36,#52));
#28=IFCLOCALPLACEMENT(#22,#27);
#27=IFCAXIS2PLACEMENT3D(#24,#25,#26);
#24=IFCCARTESIANPOINT((2.,1.,0.));
#25=IFCDIRECTION((0.,0.,1.));
#26=IFCDIRECTION((1.,0.,0.));

/* first representation - wall axis */
#36=IFCSHAPEREPRESENTATION(#11,'Axis','GeometricSet',(#35));
#35=IFCTRIMMEDCURVE(#32,(IFCPARAMETERVALUE(0.),#33),(IFCPARAMETERVALUE(1.),#34),.T.,
    .CARTESIAN.);
#32=IFCLINE(#29,#31);
#29=IFCCARTESIANPOINT((0.,0.));
#31=IFCVECTOR(#30,2.8);
#30=IFCDIRECTION((1.,0.));
#33=IFCCARTESIANPOINT((0.,0.));
#34=IFCCARTESIANPOINT((2.8,0.));

/* second representation - wall extrusion of a profile */
#52=IFCSHAPEREPRESENTATION(#11,'Body','SweptSolid',(#50));
#50=IFCEXTRUDEDAREASOLID(#44,#48,#49,2.8);
#44=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#43);
#43=IFCPOLYLINE((#38,#39,#40,#41,#42));
#38=IFCCARTESIANPOINT((0.,0.1));
#39=IFCCARTESIANPOINT((0.,-0.1));
```

```
#40=IFCCARTESIANPOINT((2.8,-0.1));
#41=IFCCARTESIANPOINT((2.8,0.1));
#42=IFCCARTESIANPOINT((0.,0.1));
#48=IFCAXIS2PLACEMENT3D(#45,#46,#47);
#45=IFCCARTESIANPOINT((0.,0.,0.));
#46=IFCDIRECTION((0.,0.,1.));
#47=IFCDIRECTION((1.,0.,0.));
#49=IFCDIRECTION((0.,0.,1.));

/* styled representation – presentation for axis */
#52=IFCSTYLEDREPRESENTATION(#12,'AxisStyle','',(#107));
#107=IFCANNOTATIONCURVEOCCURRENCE(#35,(#108),'');
#108=IFCPRESENTATIONSTYLEASSIGNMENT((#109));

/* styled representation – presentation for body */
#62=IFCSTYLEDREPRESENTATION(#13,'BodyStyle','',(#120));
#120=IFCANNOTATIONSURFACEOCCURRENCE(#50,(#121),'');
#121=IFCPRESENTATIONSTYLEASSIGNMENT((#122,#132));

/* curve style for axis – maybe shared */
#109=IFCCURVESTYLE($,#110,$,#115);
#110=IFCCURVESTYLEFONTANDSCALING('',#111,0.01);
#111=IFCCURVESTYLEFONT('Strichpunkt',(#112,#113));
#112=IFCCURVESTYLEFONTPATTERN(6.27,6.26);
#113=IFCCURVESTYLEFONTPATTERN(0.02,6.26);
#114=IFCCURVESTYLEFONTPATTERN(6.27,0.);
#115=IFCDRAUGHTINGPREDEFINEDCOLOUR('red');

/* curve style for body – maybe shared */
#132=IFCCURVESTYLE($,#133,$,#134);
#133=IFCDRAUGHTINGPREDEFINEDCURVEFONT('continuous');
#134=IFCCOLOURRGB(0.7019607843,0.7019607843,0.7019607843);

/* surface style for body – maybe shared */
#122=IFCSURFACESTYLE(.BOTH.,$,$,(#123));
#123=IFCSURFACESTYLESHADING(#124);
#124=IFCCOLOURRGB(0.9686274509,0.7764705882,0.);

/* assignment to layer structure */
#221=IFCPRESENTATIONLAYERASSIGNMENT('A-WALL-AXIS','Wall Axis',(#35));
#222=IFCPRESENTATIONLAYERASSIGNMENT('A-WALL-EXT','External Walls (Body)',(#50));

/* associated material and wall layering information */
#57=IFCMATERIALLAYERSETUSAGE(#56,.AXIS2.,.POSITIVE.,-0.1);
#56=IFCMATERIALLAYERSET((#55),'Single Layer Concrete');
#55=IFCMATERIALLAYER(#54,0.2,.F.);
#54=IFCMATERIAL('Single Layer Concrete');

/* different representation contexts for multiple representations */
/* master geometric representation context */
#11=IFCGEOMETRICREPRESENTATIONCONTEXT('Model','',3,0.0000001,#10,$);
#10=IFCAXIS2PLACEMENT3D(#7,#8,#9);
#7=IFCCARTESIANPOINT((0.,0.,0.));
#8=IFCDIRECTION((0.,0.,1.));
#9=IFCDIRECTION((1.,0.,0.));

/* geometric representation sub context for the single line axis representation */
#12=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('','Graph',*,*,*,*,#11,$,.GRAPH_VIEW.,$);
#13=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('','Model',*,*,*,*,#11,$,.MODEL_VIEW.,$);
```

9.2.2 Geometric representation and presentation of annotations

The new IFC2x2 entity *IfcAnnotation* is used to encapsulate all “free” or “unassigned” geometries, that not directly relate to a semantic object (i.e. any specific subtype of *IfcElement* or any *IfcProxy*).

Examples are free text and annotations, dimensions, etc. not associated to a single object

The *IfcAnnotation*, as subtype of *IfcProduct*, does have the same units of functionality as other *IfcElement*’s but with some practical restrictions (e.g. no typing, no decomposition). All instances of *IfcAnnotation* can therefore perform:

- "Identification" and "Change Management", see section 3.1.
- "External References" to documents, classifications and libraries, see section 10.4.
- "Placement" and "Shape Representation", see section 2, refined as spatial structure element specific shape representation (also referred to as geometry use cases) within this section.
- "Element in Space Containment", see section 3.

The *IfcAnnotation* can be seen as a container object, that includes a single or many annotations, that are placed within the local coordinate system of a spatial structure element (usually the building story), and that have one to many representations and presentations of its content, which could be specific to a certain scale or projection, as expressed by the *IfcGeometricRepresentationSubContext*.

The annotations, that cannot be assigned to object representations, are group according to a spatial structure. Usually it would be *IfcBuildingStorey*, or *IfcBuilding*, or *IfcSite*. The local placement of the *IfcAnnotation* is then relative to these spatial structure elements, or to the global world coordinate system. If a 2D local placement is used, it is placed within the XY plane of the referenced coordinate system.

Example: The following simple example shows two geometric representation items (a line and a circle) that have curve presentation properties (color and line type)and are grouped an annotation of the building storey. Both color and line type are given explicitly (i.e. not through layer setting).

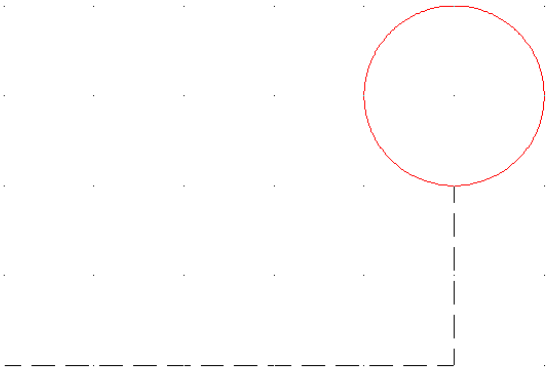


Figure 129 : Example of annotation object

The examples shows the assignment of the "free lines" to an annotation object for the whole building storey. The geometric representation items are grouped as a shape representation (for IFC2x upward compatibility) and the presentation information is given as an IfcStyledRepresentation. In this example, only one presentation (applicable to the model) is given, which is independent to scale and projection.

```
/* annotation object with local placement and containment */
#100=IFCANNOTATION('0vZnM3yZjD48ZdN7dQ17oA',#2,$,$,$,#102,$);
#101=IFCRELCONTAINEDINSPATIALSTRUCTURE('2$WrbX8KbDhRdQlWgOqya4',#2,$,$,(#100),#31);
#102=IFCLOCALPLACEMENT(#32,#103);
#103=IFCAXIS2PLACEMENT3D(#104,$,$);
#104=IFCCARTESIANPOINT((0.,0.,0.));

/* shape representation of annotation */
#110=IFCSHAPEREPRESENTATION(#21,'','',(#112,#116));
#112=IFCPOLYLINE((#113,#114,#115));
#113=IFCCARTESIANPOINT((0.,3.,0.));
#114=IFCCARTESIANPOINT((5.,3.,0.));
#115=IFCCARTESIANPOINT((5.,5.,0.));
#116=IFCCIRCLE(#117,1.);
#117=IFCAXIS2PLACEMENT3D(#118,$,$);
#118=IFCCARTESIANPOINT((5.,6.,0.));

/* presentation of annotation -general for model- */
#120=IFCSTYLEDREPRESENTATION(#23,'','',(#121,#123));
#121=IFCANNOTATIONCURVEOCCURRENCE(#112,(#122),'');
#122=IFCPRESENTATIONSTYLEASSIGNMENT((#130));
#123=IFCANNOTATIONCURVEOCCURRENCE(#116,(#124),'');
#124=IFCPRESENTATIONSTYLEASSIGNMENT((#140));
```

```
/* curve styles for polyline and circle */
#130=IFCCURVESTYLE('',#131,$,#132);
#131=IFCDRAUGHTINGPREDEFINEDCURVEFONT('dashed');
#132=IFCDRAUGHTINGPREDEFINEDCOLOUR('black');
#140=IFCCURVESTYLE('',#141,$,#142);
#141=IFCDRAUGHTINGPREDEFINEDCURVEFONT('continuous');
#142=IFCDRAUGHTINGPREDEFINEDCOLOUR('red');

/* assignment to layer structure */
#201=IFCPRESENTATIONLAYERASSIGNMENT('0','',(#112,#116));

/* project definition */
#1=IFCPROJECT('0q5sIyGrnDjvVsgb$_5xee',#2,'TestProject','Testing','','IFC 2x2 Processor
Development Test file','',(#21),#3);
#40=IFCRELAGGREGATES('1ufOV9ozf9hAHd1ewIoSxA',#2,'Testrelation','Default
Site',#1,(#41));
#41=IFCSITE('1EksLiLvv4BRGS0LTK2cuG',#2,'Testsite','Default
Site',$,#42,$,'',.ELEMENT.,$,$,0.,'',$);
#42=IFCLOCALPLACEMENT($,#43);
#43=IFCAXIS2PLACEMENT3D(#44,#45,#46);
#44=IFCCARTESIANPOINT((0.,0.,0.));
#45=IFCDIRECTION((0.,0.,1.));
#46=IFCDIRECTION((1.,0.,0.));
#30=IFCRELAGGREGATES('2IjqaFItn6ivAK75iq_317',#2,'Testrelation','Default
Building',#41,(#31));
#31=IFCBUILDING('0ZI8XAX9v7ax9yB28Rf6mc',#2,'Testbuilding','Default
Building',$,#32,$,'',.ELEMENT.,0.,0.,$);
#32=IFCLOCALPLACEMENT(#42,#33);
#33=IFCAXIS2PLACEMENT3D(#34,#35,#36);
#34=IFCCARTESIANPOINT((0.,0.,0.));
#35=IFCDIRECTION((0.,0.,1.));
#36=IFCDIRECTION((1.,0.,0.));

/* master geometric representation context */
#21=IFCGEOMETRICREPRESENTATIONCONTEXT('Model','',3,0.0000001,#22,$);
#22=IFCAXIS2PLACEMENT3D(#27,#28,#29);
#27=IFCCARTESIANPOINT((0.,0.,0.));
#28=IFCDIRECTION((0.,0.,1.));
#29=IFCDIRECTION((1.,0.,0.));

/* geometric representation sub context for the general presentation */
#23=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('','Model',*,*,*,*,#21,$,.MODEL_VIEW.,$);
```

9.2.3 Individual presentation occurrences

This section discusses how the styled items and annotation occurrences are defined depending on their style assignment. The current IFC2x2 differentiates between:

- curve styles
- symbol styles
- fill area styles
- text styles
- surface styles

Each style is introduced in a sub section. The style definition always contains an geometric representation item (the geometric instance), an appearance definition (the shared style information) and a style body (the relation between the two).

9.2.3.1 Curve style and annotation curve occurrence

An *IfcStyledItem* being an *IfcAnnotationCurveOccurrence* is a geometric curve definition (see subtypes of *IfcCurve*) together with an *IfcCurveStyle*. An *IfcCurveStyle* can also be associated with geometric surface definitions (to define the appearance of boundary curves) and with geometric solid definitions (to define the appearance of solid edges).

In a simple example, a line, given by *IfcPolyline*, shall have a line style. The Figure 129 shows the definition needed to provide for a styled line. The additional presentation entities allow for the presentation for the line color (blue, red, etc.), the line style (continuous, dashed, etc.), and the line width. The four attributes of *IfcCurveStyle* shall be interpreted as:

- *Name*: the name of the curve style, if part of a curve style library. If no significant name is available, an empty string, ‘’, shall be exchanged.
- *CurveFont*: the curve font appearance, since only the model space presentation is in scope, the actual length of segments within a curve font may have to be adjusted by a scaling factor. The curve font can be given as:
  - explicit setting of an individual curve font: the font is then given by assigning a value to the *CurveFont* attribute, this value could be:
    - an *IfcDraughtingPreDefinedCurveFont* for standard fonts currently including 'continuous', 'chain', 'chain double dash', 'dashed', and 'dotted'. A table within the IFC specification gives the lengths for the visible and invisible segments of each font.
    - an *IfcCurveStyleFont* for user or system defined fonts, the length of the visible and invisible segments are explicitly exchanged as values of *IfcPositiveLengthMeasure* (i.e. given by the global length unit).
    - an *IfcCurveStyleFontAndScaling* for curve fonts (being either *IfcCurveStyleFont* or *IfcDraughtingPreDefinedCurveFont*) that have a scale factor, this allows to share the same curve font definition, if only the scale varies, but the proportion remains the same.
  - setting of the curve font by layer settings: the *CurveFont* attribute shall be undetermined (nil) and the representation item (the *IfcGeometricRepresentationItem*) shall be an *AssignedItems* of an *IfcPresentationLayerAssignmentWithStyle* having a value given for the attribute *LayerCurveFont*. Then the curve font given to the layer shall be taken.
  - default setting ‘continuous’ shall be used, if no value is given for the *CurveFont* attribute and either no layer information, given by *IfcPresentationLayerAssignmentWithStyle* is available, or the *LayerCurveFont* attribute of *IfcPresentationLayerAssignmentWithStyle* is not provided.
- *CurveColour*: the curve colour appearance to be used to display the curve, The curve colour can be given as:
  - explicit setting of an individual curve color: the color is then given by assigning a value to the *CurveColour* attribute, this value could be:
    - an *IfcDraughtingPreDefinedColour* for standard colours currently including 'black', 'red', 'green', 'blue', 'yellow', 'magenta', 'cyan', and 'white'. A table within the IFC specification gives the RGB values for each colour name.
    - an *IfcColourRgb* for user or system defined colours using the RGB colour model, the individual values for the red, green, and blue component are explicitly exchanged as values of *IfcNormalisedRatioMeasure* (Note: the often used values 0..255 have to be transformed in normalised values 0..1).
  - setting of the curve colour by layer settings: the *CurveColour* attribute shall be undetermined (nil) and the representation item (the *IfcGeometricRepresentationItem*) shall be an *AssignedItems* of an *IfcPresentationLayerAssignmentWithStyle* having a value given for the attribute *LayerCurveColour*. Then the curve colour given to the layer shall be taken.
  - default setting ‘black’ shall be used, if no value is given for the *CurveFont* attribute and either no layer information, given by *IfcPresentationLayerAssignmentWithStyle* is available, or if the *LayerCurveColour* attribute of *IfcPresentationLayerAssignmentWithStyle* is not provided.
- *CurveWidth*: the curve width appearance to be used to display the curve, since only model space representation is in scope, the curve width may not be shown (but might be used for later projections onto paper space). The curve width can be given as:
  - explicit setting of an individual curve width: the width is then given by assigning a value to the *CurvWidth* attribute, this value could be either given within the global length unit (as *IfcPositiveLengthMeasure*) or with explicit units (as *IfcMeasureValueWithUnit* – usually mm is used for metric measures).
  - setting of the curve width by layer settings: the *CurveWidth* attribute shall be undetermined (nil) and the representation item (the *IfcGeometricRepresentationItem*) shall be an *AssignedItems* of an *IfcPresentationLayerAssignmentWithStyle* having a value given for the attribute *LayerCurveWidth*. Then the curve width given to the layer shall be taken.
  - if no value is given for the *CurveFont* attribute and either no layer information, given by *IfcPresentationLayerAssignmentWithStyle* is available, or if the *LayerCurveColour* attribute of *IfcPresentationLayerAssignmentWithStyle* is not provided, then a system default shall be used for curve width.

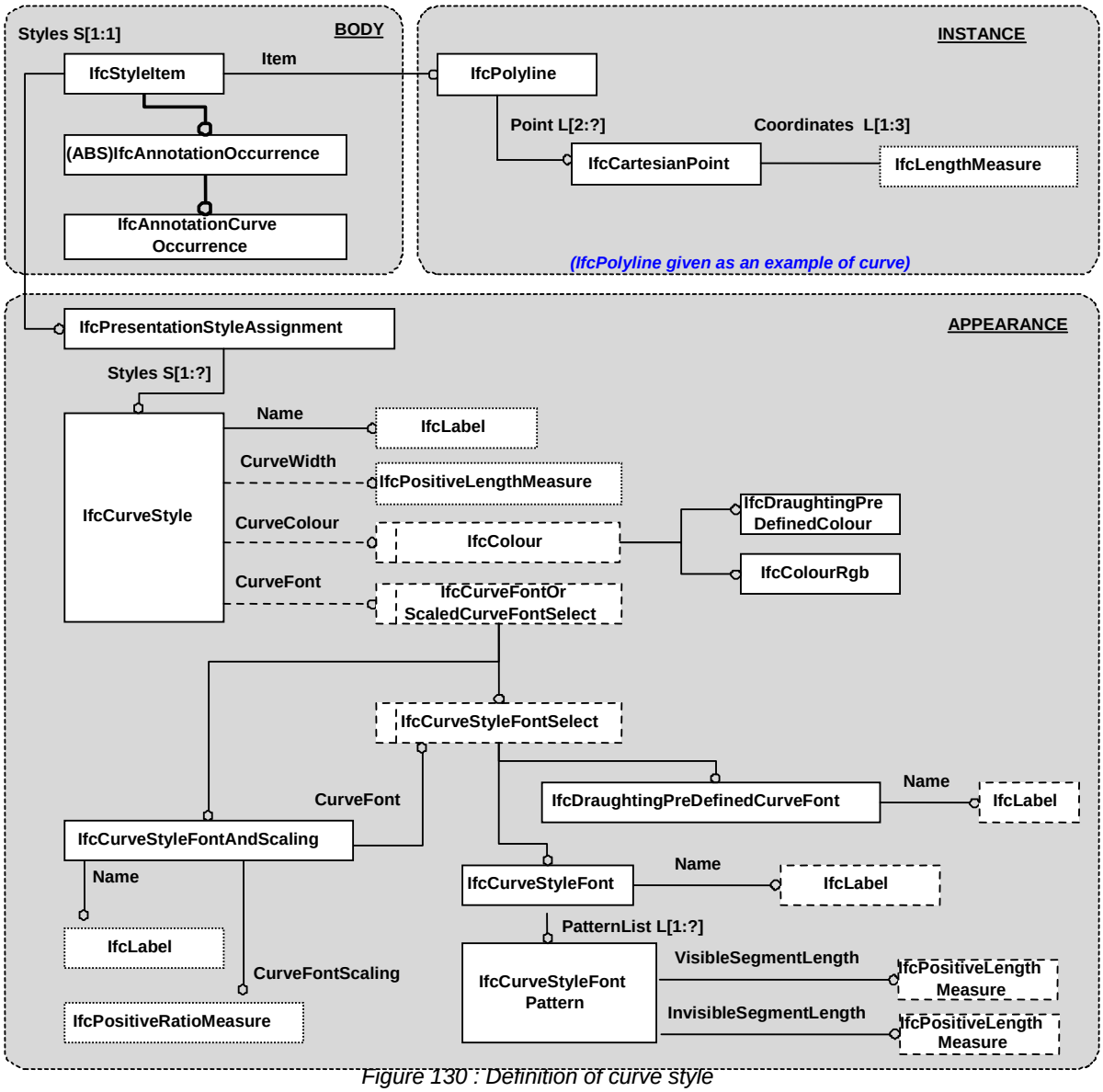


Figure 130 : Definition of curve style

9.2.3.2 Symbol style and annotation symbol occurrence

9.2.3.3 Fill area style and annotation fill area occurrence

9.2.3.4 Text style and annotation text occurrence

9.2.3.5 Surface style and annotation surface occurrence

An *IfcStyledItem* being an *IfcAnnotationSurfaceOccurrence* is a geometric surface definition, face based or shell based representation, or solid model, together with an *IfcSurfaceStyle*. It is applicable to:

- *IfcSurface*
- *IfcSolidModel*
- *IfcShellBasedSurfaceModel*
- *IfcFaceBasedSurfaceModel*

An *IfcCurveStyle* can also be associated with geometric surface definitions (to define the appearance of boundary curves) and with geometric solid definitions (to define the appearance of solid edges).

9.2.3.5.1 Surface style shading

The shading style is the most simple surface style that can be defined within IFC. It includes the definition of a surface color and optionally a definition of the curve color which is used for the color appearance of edges, if the shading mode selected includes visible edges. The necessary elements for enabling the shading definitions are explained in Figure 130.

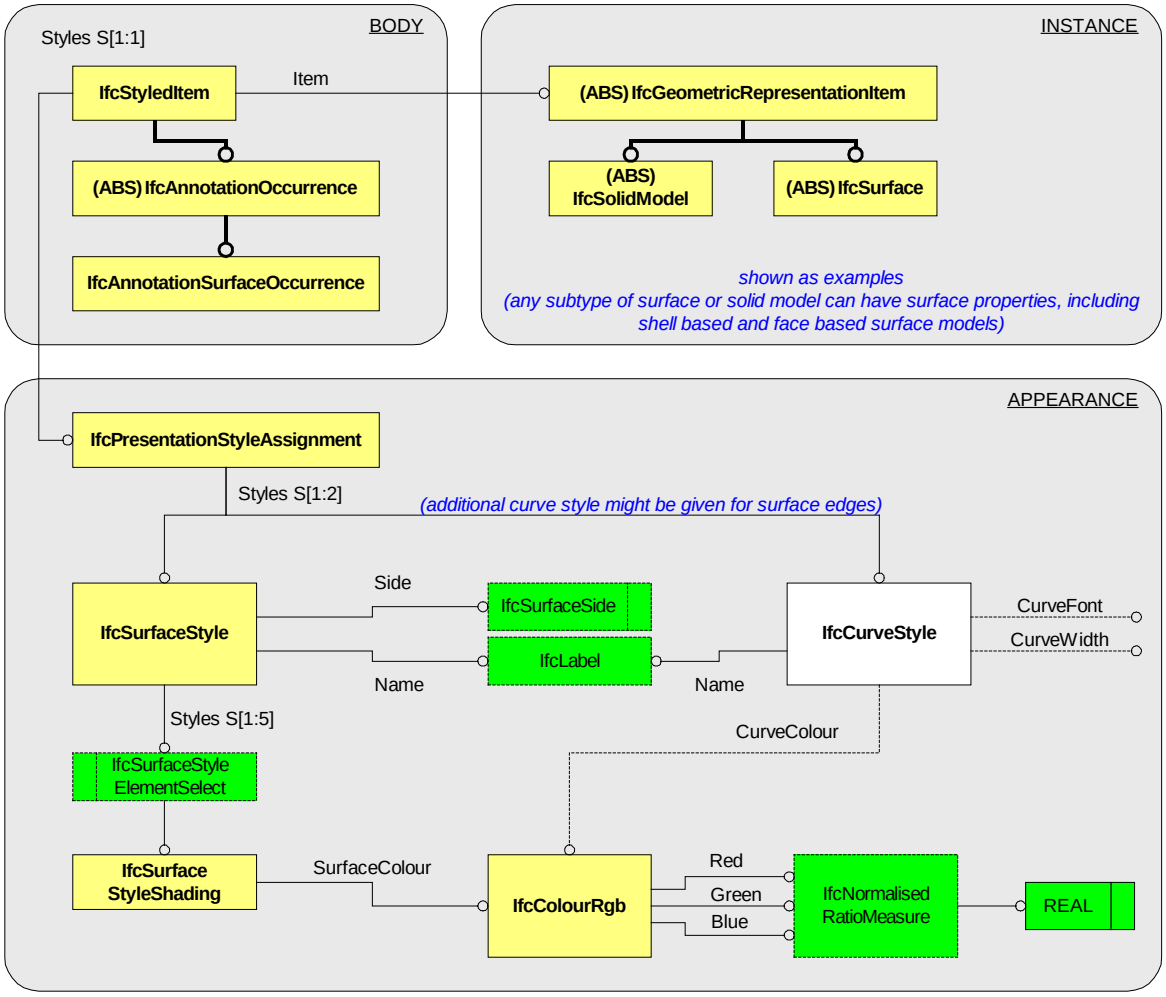


Figure 131 : Definition of surface style shading

10 Properties of Elements

10.1 Property Definition

The IFC Schema comprises a set of well defined ways of breaking down information into classes and the structure of information that defines an objects (which is an instance of a class in use). The information structures provide a formal specification of attributes that belong to classes and define how data exchange and sharing using ISO 10303 parts 21 and 22 or other encoding method will be achieved.

However, there are many types of information that users might want to exchange that are not currently included within the IFC Model.

For this purpose the IFC Model provides the Property Definition mechanism (part of which is within the *IfcKernel* schema with the remainder being within the *IfcPropertyResource* schema). Property Definition is a generic mechanism that allows model users and developers to define, connect and use data-driven, expandable properties with objects

Property Definitions can be either:

- type defined and shared among multiple instances of a class, or
- type defined but specific for a single instance of a class, or
- extended definitions that are added by the end users.

10.1.1 Extended Concept of Property Definition

The concepts provided by *IfcPropertyDefinition* and its subtypes are further described here. It allows for:

- Relating of an object type, for which a set of properties is defined. This is done though assigning an *IfcTypeObject* (through the *IfcRelDefinesByType* relationship) to a single or multiple object occurrences.

  For instance, there may be a class called IfcFan within the statically defined model[19]. However, the different types of fan that may exist (single stage axial, multi-stage axial, centrifugal, propeller etc.) are not in the statically defined model. These may be declared as types of the fan through a type relationship attached to the IfcFan class. Each type of fan that could be defined in IFC is included in an enumeration of fan types, or maybe extended by a type designation captured in the general ObjectType string attribute. The value that these attributes take defines the IfcTypeObject that is assigned.

- Sharing a standard set of values defined in an *IfcPropertySet* (or any other subtype of *IfcPropertySetDefinition*) across multiple instances of that class (through the *IfcRelDefinesByProperties* relationship).

  For instance, a standard range of properties with known values might be defined for the maintenance of centrifugal fans. These properties will be applied to every centrifugal fan by attaching the same instance of IfcPropertySet to all instances of IfcFan through the same 1:N relationship instance IfcRelDefinesByProperties.

- Defining different property values within a private copy of the *IfcPropertySet* for each instance of that class.

  For instance, all centrifugal fans deliver a volume of air against a known resistance to airflow. Although these properties are assigned to every centrifugal fan, the values given to them differ for every instance. There are two options to attach these private properties:

  - If they overlap with properties within the shared property sets, i.e. they redefine a commonly shared property value, the properties should be assigned as overriding properties through the *IfcRelOverridesProperties* relationship.

[19] The term ‘statically defined model’ is used to refer to the EXPRESS specification part of the IFC Model. Use of the term ‘dynamic’ or ‘dynamically defined model’ is restricted to those parts of the IFC Model that are not formally defined in EXPRESS or extensions to the IFC Model that are not documented in an IFC release.

- If they add to the commonly shared properties within the assigned property sets, then they should be added by another instance of *IfcRelDefinesByProperties*, assigning the specific single property set.

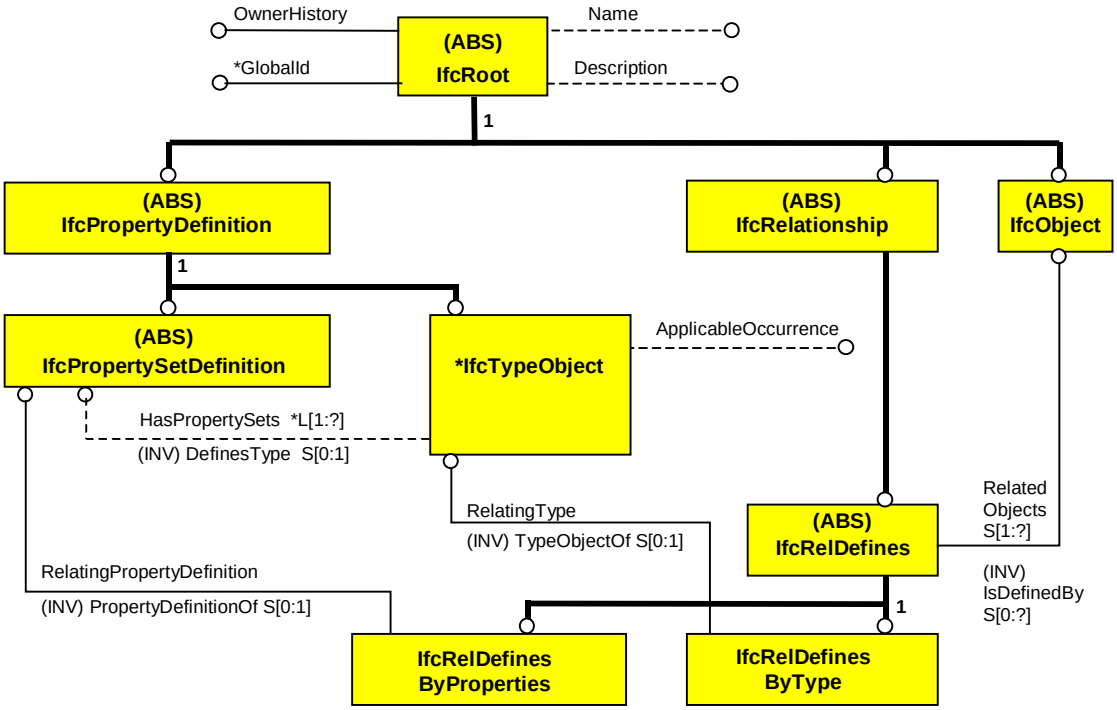


Figure 132 : Definition of Property Definitions

10.1.2 Extended concept of Property Set Definition

The concepts of *IfcPropertySetDefinition* and its subtypes are further described here.

The *IfcPropertySetDefinition* is an abstract supertype of property sets that can be used within the IFC Model. These include both the dynamically defined property sets declared through the *IfcPropertySet* class and the statically defined property sets such as the *IfcManufactureInformation* class, which represents a subtype of *IfcPropertySetDefinion* defined elsewhere in the IFC2x model. Both types of property set definitions can be distinguished as:

- **Statically defined properties**, they define properties for which a entity definition exist within the IFC model. The semantic meaning of each statically defined property is declared by its entity type and the meaning of the properties is defined by the name of the explicit attribute.
- **Dynamically extendable properties**, they define properties for which the IFC model only provides a kind of "meta model", to be further declared by agreement. This means no entity definition of the properties exists within the IFC model. The declaration is done by assigning a significant string value to the "Name" attribute of the entity as defined in the entity *IfcPropertySet* and at each *IfcProperty*, referenced by the property set.

10.1.2.1 Property Set Definition Attachment

Both, statically and dynamically defined property sets are attached to an object using the relationship class *IfcRelDefinesByProperties*. This allows for the object and the property definition to exist independently and for the *IfcPropertySetDefinition* to be attached to the object when required. An advantage of using the relationship class is that the object does not contain any references to property set definitions if none needed.

Use of the relationship class also allows the property set definition to be assigned to one or many objects. That is, many objects may share a single property set definition with common values if required.

For instance, if there are 7 chairs (instances of the IfcFurniture class) that are all exactly the same, they can all share a reference to the same IfcPropertySetDefinition that defines information about the type of upholstery, color etc.

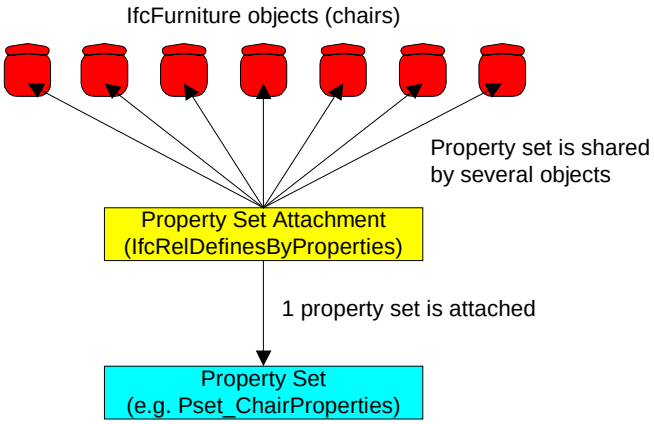


Figure 133 : Example of Property Set Atachment

Both the *IfcObject* and the *IfcPropertySetDefinition* have inverse attributes to *IfcRelDefinesByProperties*. Whilst these cannot be seen in an IFC exchange file formatted according to ISO 10303 Part 21, they can be used by an application. For instance, an object may be defined by many property definition assignments and these can be determined by the inverse attribute.

10.1.2.2 Overriding of Property Definitions

A property set may be assigned to many objects. However, it may be the case the value of some properties in a subset of the objects may vary whilst others remain constant. Rather than defining and assigning new property sets, the IFC Schema provides the capability to override those properties whose values change. This is done by the assignment of an overriding property set that contains new values for those that have varied.

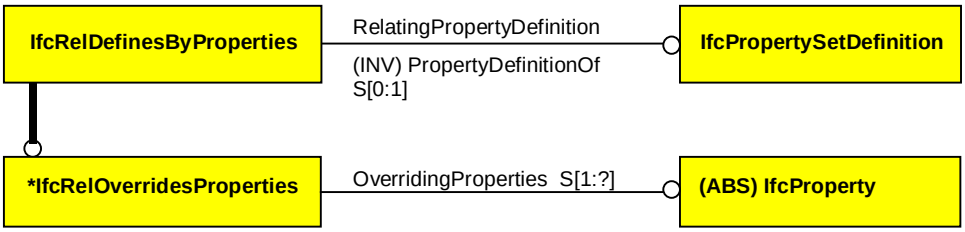


Figure 134 : Definition of IfcRelOverridesProperties

Example:

Figure 134 shows a set of objects (here chairs) that are to be assigned a property set that contains properties. Of this set of chairs, there is one chair which has a different value for one property (here color). Therefore the standard value within the shared property set is overridden by the specific value for that individual chair. The value of all other properties in the subset remain unchanged, as do the values of all properties assigned to the other objects.

Note that there must be a total correspondence between the names of the properties in the set of overriding properties and the names of the properties whose values are to be changed in the base property set. In addition the inherited attribute *RelatingPropertyDefinition* points to the property set which values are overridden.

The pointer to the actual original property set is necessary to avoid redundancies, if the overriding properties may appear (by name) at more than on property set, attached to the object.

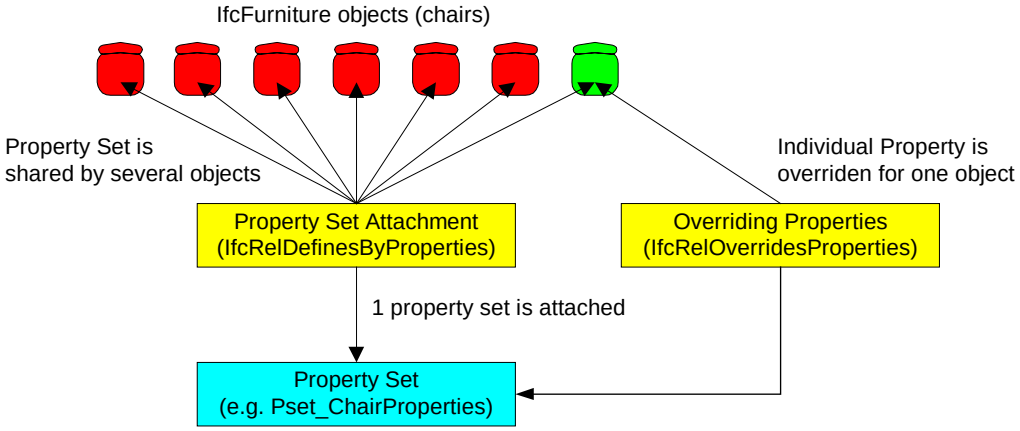


Figure 135 : Example of Overriding Properties

10.1.3 Extended concept of Type Object

The concepts of *IfcTypeObject* and its subtypes are further described here.

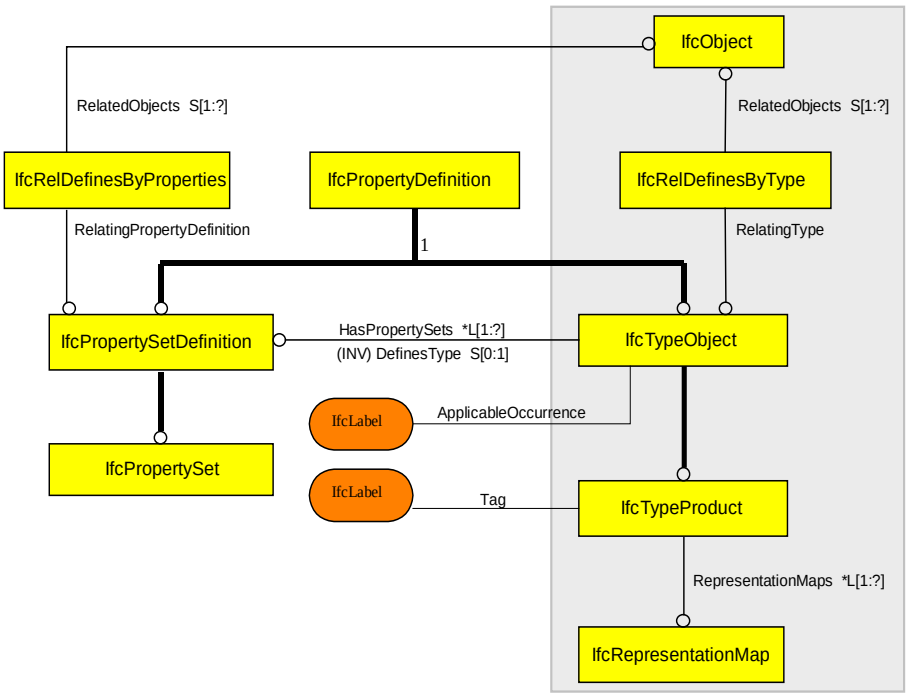


Figure 136 : Definition of Type Object and Type Product

An *IfcTypeObject* provides the means for grouping together a number of property sets that commonly work together. This is in preference to a situation where every property set is individually assigned to an object (although it is still possible to assign property sets individually). An *IfcTypeObject* may act as the container for a list of property sets (property set definitions) where the list is ordered (each property set is in the same relative location for each instance of the *IfcTypeObject*) and there is no duplication of property sets. An inverse relationship between *IfcPropertySetDefinition* and *IfcTypeObject* provides for the possibility of relating the definition to an object type within which it is contained.

- Each *IfcTypeObject* has a name (inherited from *IfcRoot*) that identifies it. This could, for instance, be used in the context of library information as a means of acquiring several property sets at the same time and assigning them to an object via the *IfcTypeObject*.

- The *Description* attribute (also inherited from *IfcRoot*) may be used to provide further, human readable, narrative information about the object type.
- The *ApplicableOccurrence* attribute may be used to constrain the *IfcTypeObject* with regard to the class within the IFC Model to which it can be assigned. This acts in the same manner as for type defined classes in previous releases of the IFC Model.

10.1.3.1 Type Object Attachment

The *IfcTypeObject* instance is attached to an object using the relationship class *IfcRelDefinesByType*. This allows for the object and the *IfcTypeObject* to exist independently and for the *IfcTypeObject* to be attached to the object when required. An advantage of using the relationship class is that the object does not contain any references to property set definitions if none are needed.

Use of the relationship class also allows the *IfcTypeObject* to be attached to one or many objects. That is, many objects may share a single *IfcTypeObject* with its contained property set definitions if required.

For instance, if there are 7 chairs (instances of the IfcFurniture class) that are all exactly the same, and there are multiple property sets that act together within an IfcTypeObject to describe the chair they can all share a reference to the same IfcTypeObject.

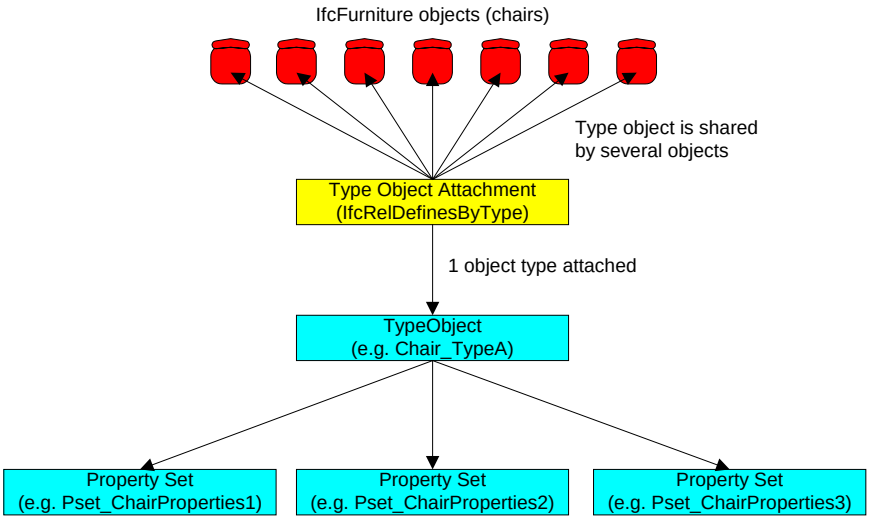


Figure 137 : Example of Type Object Attachment

Example shows the type object containing all property sets and its attachment to the chairs.

```
#1=IFCTYPEOBJECT('abcdefghijklmnopqrst02', #2, 'Type_Chair', $, $, (#1001,#1002,#1003));

/* Property Sets contained by the Type Object */
#1001=IFCPROPERTYSET('abcdefghijklmnopqrst03', #2, 'Pset_ChairAsDesigned', $, (#1101));
#1002=IFCPROPERTYSET('abcdefghijklmnopqrst04', #2, 'Pset_ChairAsInstalled', $, (#1102));
#1003=IFCPROPERTYSET('abcdefghijklmnopqrst05', #2, 'Pset_ChairAsUsed', $, (#1103));
#1101=IFCPROPERTYSINGLEVALUE('Heigth',$,IFCLENGTHMEASURE(1.000000E+00),$);
#1102=IFCPROPERTYSINGLEVALUE('Heigth',$,IFCLENGTHMEASURE(1.500000E+00),$);
#1103=IFCPROPERTYSINGLEVALUE('Heigth',$,IFCLENGTHMEASURE(1.500000E+00),$);

/* assignment of type object to the chairs */
#2000=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst06', #2, $, $,
      (#2100,#2101,#2102,#2103,#2104,#2105,#2106), #1);
#2100=IFCFURNITURE('abcdefghijklmnopqrst10', #2, $, $, $, $, $, $);
#2101=IFCFURNITURE('abcdefghijklmnopqrst11', #2, $, $, $, $, $, $);
#2102=IFCFURNITURE('abcdefghijklmnopqrst12', #2, $, $, $, $, $, $);
#2103=IFCFURNITURE('abcdefghijklmnopqrst13', #2, $, $, $, $, $, $);
#2104=IFCFURNITURE('abcdefghijklmnopqrst14', #2, $, $, $, $, $, $);
#2105=IFCFURNITURE('abcdefghijklmnopqrst15', #2, $, $, $, $, $, $);
#2106=IFCFURNITURE('abcdefghijklmnopqrst16', #2, $, $, $, $, $, $);
```

10.1.4 Extended concept of Type Product

The *IfcTypeProduct* class enables a property definition to have one or more shape representations through the *RepresentationMaps* attribute. This attribute has the type *IfcRepresentationMap* that is the class in the *IfcGeometryResource* schema that defines a group of geometrical objects that can act together and be assigned to multiple objects in the manner of a symbol or block in a CAD system.

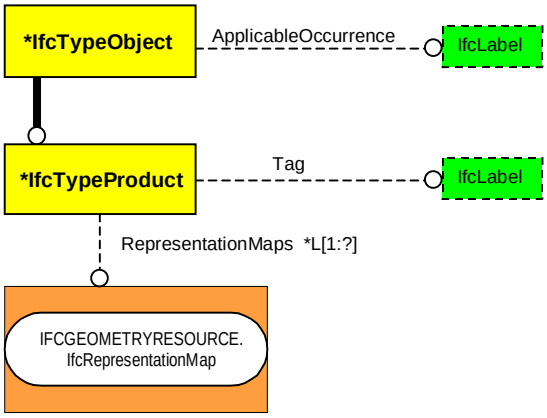


Figure 138 : Definition of IfcTypeProduct

Example:

All instances of IfcFurniture (here chairs) does not only share the same set of properties (by a common IfcTypeObject), but also the same geometric representation. Therefore the subtype IfcTypeProduct is used, which allows the sharing of the same representation maps. A set of representation maps can be seen as a multi-view block, i.e. a block definition, that includes several blocks for different representation views. For example a 2D representation of the chair and a 3D B-rep representation of the chair.

Each instance of the IfcFurniture would then have its own local placement (see 9.1.2.1) and (multiple) shape representations, each holding an IfcMappedItem, which references one map of the shared representation maps.

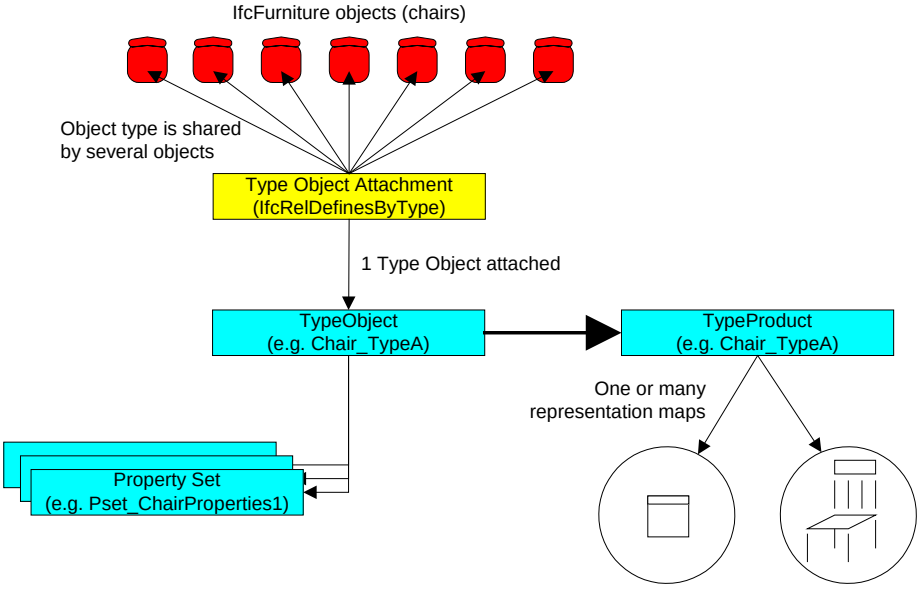


Figure 139 : Example of the IfcTypeProduct Class

An object (i.e. subtypes of *IfcProduct*) may have a representation directly referenced by the attribute *Representation* of *IfcProduct* (shown as the product route in the diagram below). The same representation may also be accessible through the attribute *RepresentationMaps* of *IfcTypeProduct* (subtype of *IfcTypeObject* and shown as the type object route in the diagram below). The reason for this duality is, that types can be exchanged without having already element instances using these

types (or in other words, blocks or libraries can be exchanged even if not already used by inserted members in the building model).

Note: Within the diagram, subtype relations are shown by the thicker, straight lines whilst the thinner, curved lines show attribute relations.

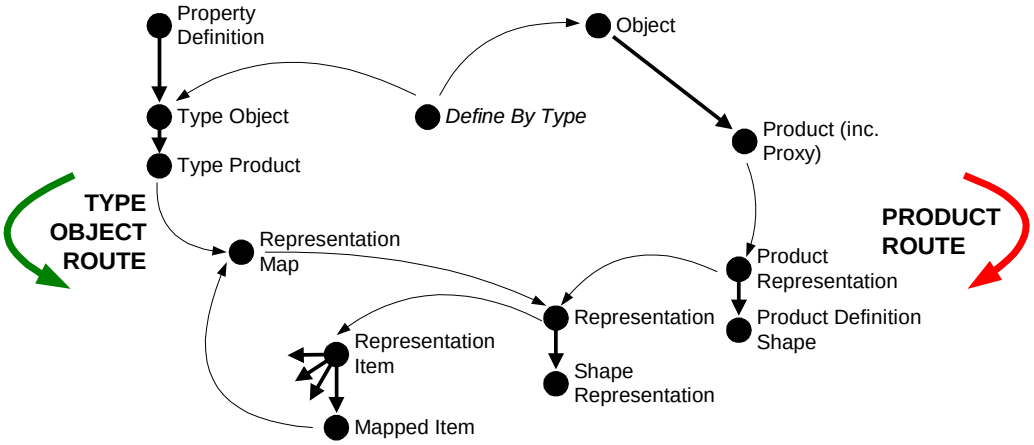


Figure 140 : Routes to representation - definition

Within a typical exchange, i.e. where geometric representation items are used to represent the shape of an element, the following links (involving particular subtypes) are used:

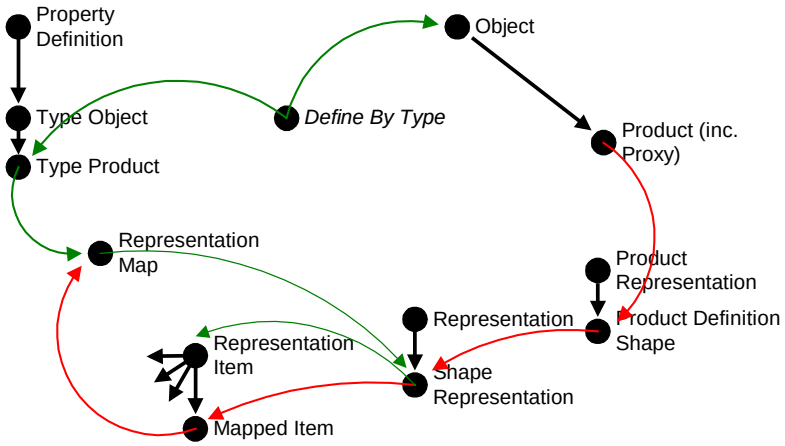


Figure 141 : Routes to representation – as used for geometric shape

Example

The example expands the previous one by adding a commonly shared representation map (the block geometry of the chair) to the chair type. Each individual chair then references the mapped item to position to chair type shape.

```
#1=IFCTYPEPRODUCT('abcdefghijklmnopqrst02', #2, 'Type_Chair', $, $, (#1001,#1002,#1003),
   (#3000), $);
#2=IFCOWNERHISTORY(#6, #7, .READWRITE., .NOCHANGE., $, $, $, 978921854);

/* common shared property sets for the type */
#1001=IFCPROPERTYSET('abcdefghijklmnopqrst03', #2, 'Pset_ChairAsDesigned', $, (#1101));
#1002=IFCPROPERTYSET('abcdefghijklmnopqrst04', #2, 'Pset_ChairAsInstalled', $, (#1102));
#1003=IFCPROPERTYSET('abcdefghijklmnopqrst05', #2, 'Pset_ChairAsUsed', $, (#1103));
#1101=IFCPROPERTYSINGLEVALUE('Heigth', $, IFCLENGTHMEASURE(1.000000E+01), $);
#1102=IFCPROPERTYSINGLEVALUE('Heigth', $, IFCLENGTHMEASURE(1.500000E+01), $);
#1103=IFCPROPERTYSINGLEVALUE('Heigth', $, IFCLENGTHMEASURE(8.500000E+00), $);

/* assignment of type to all instances of the chair */
#2000=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst06', #2, $, $, (#2103,#2104,#2105,#2106,
      #2100,#2101,#2102), #1);
#2100=IFCFURNITUREELEMENT('abcdefghijklmnopqrst10', #2, $, $, $, #3011, #3014, $);
#2101=IFCFURNITUREELEMENT('abcdefghijklmnopqrst11', #2, $, $, $, #4011, #4014, $);
#2102=IFCFURNITUREELEMENT('abcdefghijklmnopqrst12', #2, $, $, $, #5011, #5014, $);
```

```
#2103=IFCFURNITUREELEMENT('abcdefghijklmnopqrst13', #2, $, $, $, #6011, #6014, $);
#2104=IFCFURNITUREELEMENT('abcdefghijklmnopqrst14', #2, $, $, $, #7011, #7014, $);
#2105=IFCFURNITUREELEMENT('abcdefghijklmnopqrst15', #2, $, $, $, #8011, #8014, $);
#2106=IFCFURNITUREELEMENT('abcdefghijklmnopqrst16', #2, $, $, $, #9011, #9014, $);

/* common shared representation map for the chair – here simplistic bounding box */
#3000=IFCREPRESENTATIONMAP(#3003, #3005);
#3003=IFCAXIS2PLACEMENT3D(#3004, $, $);
#3004=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#3005=IFCSHAPEREPRESENTATION(#3008, 'Body', 'BoundingBox', (#3006));
#3006=IFCBOUNDINGBOX(#3007, 5.000000E-01, 4.000000E-01, 1.200000E+00);
#3007=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
#3008=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #3009, $);
#3009=IFCAXIS2PLACEMENT3D(#3010, $, $);
#3010=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));

/* shape representation for the first instance of the chair */
#3011=IFCLOCALPLACEMENT($, #3012);
#3012=IFCAXIS2PLACEMENT3D(#3013, $, $);
#3013=IFCCARTESIANPOINT((1.000000E+01, 5.000000E+00, 0.000000E+00));
#3014=IFCPRODUCTDEFINITIONSHAPE($, $, (#3015));
#3015=IFCSHAPEREPRESENTATION(#3008, 'Body', 'MappedRepresentation', (#3016));
#3016=IFCMAPPEDITEM(#3000, #3017);
#3017=IFCCARTESIANTRANSFORMATIONOPERATOR3D($, $, #3018, $, $);
#3018=IFCCARTESIANPOINT((0.000000E+00, 0.000000E+00, 0.000000E+00));
```

10.1.5 Extended concept of dynamic Property Sets

The *IfcPropertySet* class is a container that holds collections (or sets) of properties. A property set is defined externally to the statically defined IFC Model in the EXPRESS data definition language.

A number of property sets are defined and distributed with the IFC Model. Those property sets do form part of the complete IFC Model (together with the EXPRESS definition), although they are not part of the IFC2x platform.

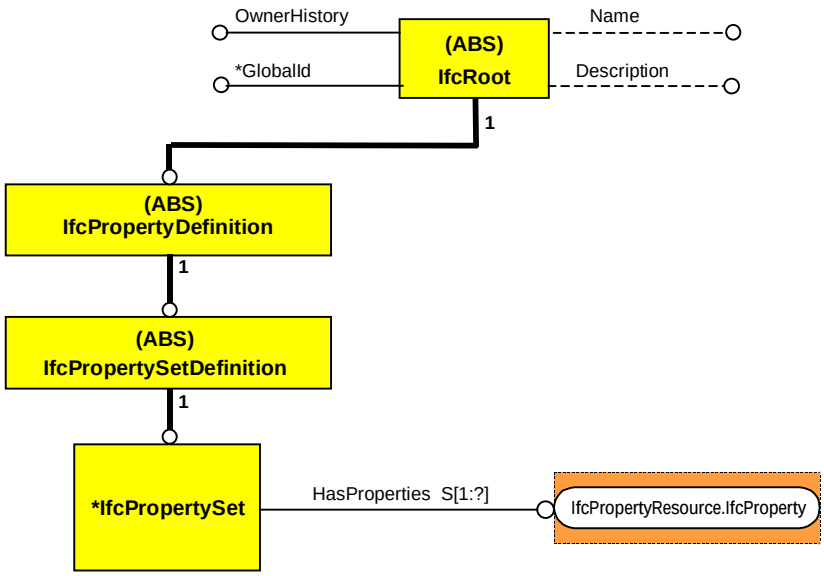


Figure 142 : Definition of IfcPropertySet

The specification of a property set defines the manner in which property information is held within a fully populated IFC exchange file or within an IFC compliant database. It can also be used to define the specification for the transport of a message containing *IfcPropertySet* information.

- An *IfcPropertySet* contains a set of properties. There must be at least one property within a property set. The *Name* attribute of all properties within the set must be different.
- An *IfcPropertySet* has a Name (inherited from *IfcRoot*). This is an identifier that is used for recognition. It is a vitally important attribute in the context of defining

industry or project standard property sets that may be used by multiple organizations.

- An *IfcPropertySet* may also have a description (inherited from *IfcRoot)* that is a human readable narrative describing some information about the property set.

10.1.6 Concept of dynamic Property Definitions

The fundamental aspect of the *IfcPropertySet* is that it contains a set of properties. It must contain at least one property and may contain as many as are necessary. Since it contains a SET of properties, the order in which the properties appear is of no significance. Only the names of the individual properties characterize their content and meaning.

A property may be either of the following (each of these types of property is further described below):

- a single value (with or without units),
- an enumeration value (with or without units),
- a bound value (with or without units),
- a list of values (with or without units) – new in IFC2x2
- a range of values (with or without units),
- an object reference,
- a complex property,

The *IfcProperty* class is the common abstraction for all Properties defined within the IFC Model. It is an abstract supertype, meaning that there is never an *IfcProperty* object itself, only objects that are a subtype of *IfcProperty*. Every instance of *IfcProperty* must have a *Name* by which it can be identified. It may have a *Description*, to further describe the meaning of the property.

> *NOTE: Within a single IfcPropertySet, all contained properties (subtypes of IfcProperty) need to have unique Name attribute values, i.e. no two properties within a property set shall have the same Name string.*

As use of the dynamic parts of the IFC Model expands, it is intended that a dictionary of standard IFC properties will be defined progressively. For the present, for those properties used in property sets that are published as part of the IFC Model, overlap in naming, definition and usage of units has been eliminated.

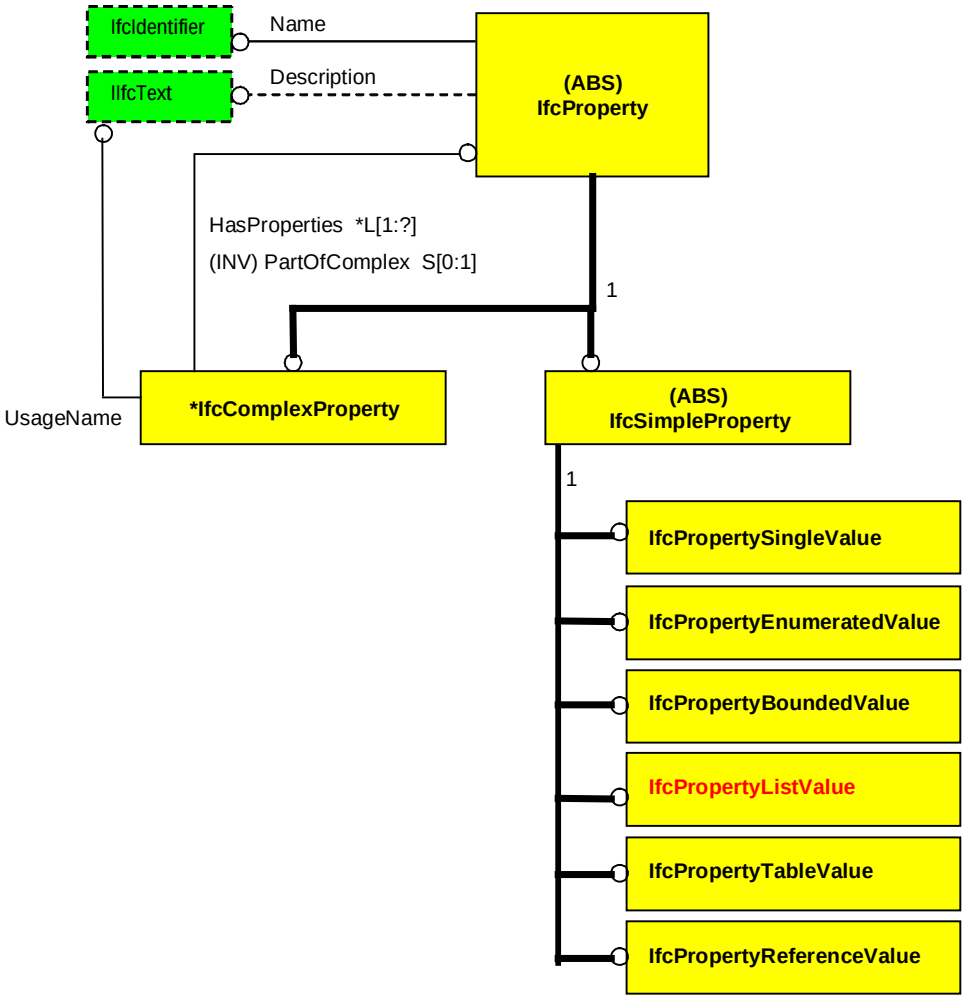


Figure 143 : Definition of IfcProperty

10.1.6.1 Concept of Property with Single Value

An *IfcPropertySingleValue* is a single property that has either a *name -- value* pair or a *name – value – unit* triplet. Provision of a *Unit* is optional. Additionally an optional *Description* may be attached.

If no *Unit* is assigned, the globally defined unit (see 3.3.2) is used as it relates to the measure type of the *NominalValue* (chosen from the *IfcValue* select type). If a *Unit* is provided, the unit for the nominal value can be defined locally (and it thereby overrides the eventually given globally defined unit). This concept applies to all subtypes of *IfcSimpleProperty*.

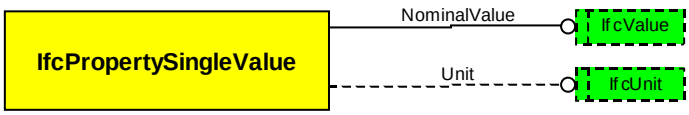


Figure 144 : The IfcPropertySingleValue Class

The *IfcPropertySingleValue* is the most used concept for exchanging property sets. If only the mandatory name – value pair is inserted it directly maps to most of the extensible attribute data of CAD systems, if a *Unit* and *Description* is to be handled as well, implementation guides may introduce special conventions for property naming in order to allow the encapsulation of a description and unit within the name tag.

> *The current implementation guide and view definitions for the coordination view recommend to not use "[", "]", ";" within a Name or Description of IfcPropertySingleValue.*

10.1.6.2 Concept of Property with Enumerated Value

An *IfcPropertyEnumeratedValue* allows for the (potentially multiple) selection of a property from a predefined list of selections. The actual value is stored by the attribute *EnumerationValues* and is selected from the list that is defined within an *IfcPropertyEnumeration* object.

The *EnumerationValues* allow for either a single choice (if the list only contains a single item) or for multiple choices. The *IfcPropertyEnumeration* instance may be referenced by the *EnumerationReference* attribute – if it is given, a where rule ensures, that the selected values are within the list of the values as specified in the *IfcPropertyEnumeration*.

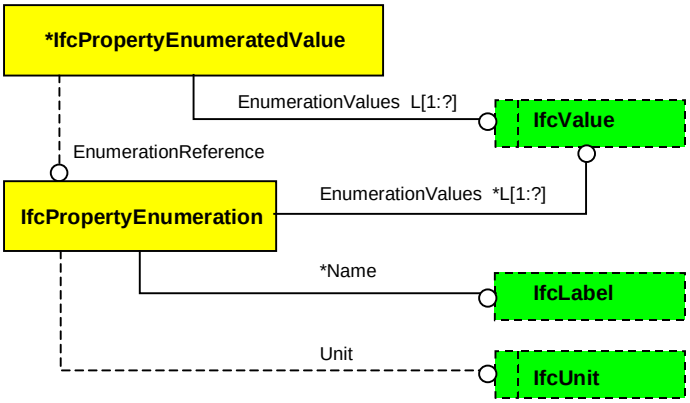


Figure 145 : Definition of IfcPropertyEnumeratedValue

The list of possible selections is defined through the use of the *IfcPropertyEnumeration* class. The actual values from which the selection is made are stored in the attribute *EnumerationValues*. Each *IfcPropertyEnumeration* has a name that must be unique to distinguish it from other instances of *IfcPropertyEnumeration* that may be used.

A unit may be assigned to an *IfcPropertyEnumeration*. This is the unit that each value in the enumeration shares. It is not possible to have values within the enumeration that have different units.

10.1.6.3 Concept of Property with Bounded Value

An *IfcPropertyBoundedValue* allows for a property whose value can be allowed to vary between an upper limit (the *UpperBoundValue* attribute) and a lower limit (the *LowerBoundValue* attribute).

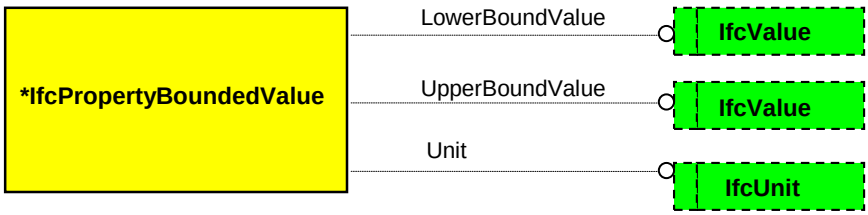


Figure 146 : Definition of IfcPropertyBoundedValue

Within IFC2x2 the LowerBoundValue and UpperBoundValue have been changed into optional attributes (had been mandatory before). Now also the exchange of open bounds is possible.

The example shows the definition of a minimum, maximum and allowable height using the features of the IfcPropertyBoundedValue.

```
#1101=IFCPROPERTYBOUNDEDVALUE('MaximumHeigth', $, IFCLENGTHMEASURE(1.0), $, $);
#1102=IFCPROPERTYBOUNDEDVALUE('MinimumHeigth', $, $, IFCLENGTHMEASURE(1.5), $);
#1103=IFCPROPERTYBOUNDEDVALUE('AllowableRangeOfHeigth', $, IFCLENGTHMEASURE(1.0),
      IFCLENGTHMEASURE(1.5), $);
```

10.1.6.4 Concept of Property with List Value

An *IfcPropertyListValue* allows for the definition of a property that is given by a list of values. This is a new entity within IFC2x2. If the *Unit* is given, it applies to all values within the list, a where rule ensures that all entries within the list are of the same measure type.

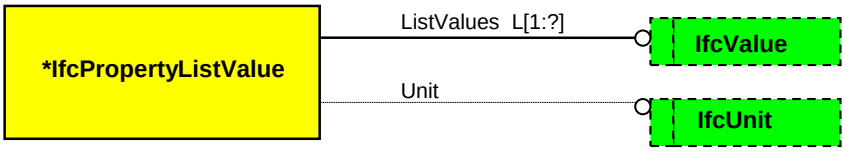


Figure 147 : Definition of IfcPropertyListValue

The example shows the instance IfcPropertyListValue keeping a list of allowable heights

```
#1101=IFCPROPERTYLISTVALUE('AllowableHeigths', $, (IFCLENGTHMEASURE(1.0),
      IFCLENGTHMEASURE(1.2), IFCLENGTHMEASURE(1.5), IFCLENGTHMEASURE(1.8)), $);
```

10.1.6.5 Concept of Property with Table Value

An *IfcPropertyTableValue* allows for the definition of a range of values where each value stored is dependant on another value. This allows for either values in a two dimensional table to be stored or approximates to the storage of a set of values that may be derived from an expression for x and y where $y = \phi(x)$

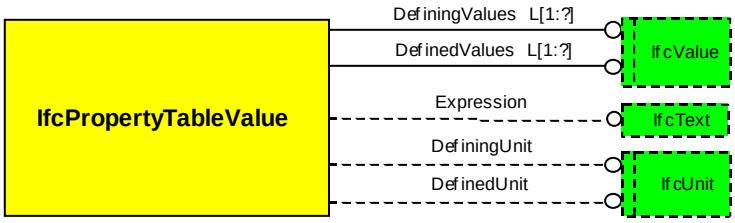


Figure 148 : Definition of IfcPropertyTableValue

Two value ranges are defined namely the *DefiningValues* and the *DefinedValues*. The *DefiningValues* are those upon which the *DefinedValues* are dependent. For example, if an *IfcPropertyTableValue* object was storing values where the expression $y = x^2$ was applicable, the table of values would be:

DefiningValues	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
DefinedValues	*1*	*4*	*9*	*16*	*25*	*36*	*49*	*64*

The Expression from which the values are derived may also be stored for reference. Using the Expression attribute is for convenience; no operations are carried out on the expression. The Units that are used for both the *DefiningValues* and the *DefinedValues* may also be defined.

10.1.6.6 Concept of Property with Reference Value

An *IfcPropertyReferenceValue* enables reference to objects whose structure is defined by the static part of the IFC Schema. This is achieved by defining a relationship to the object being referenced. This is achieved through the *IfcObjectReferenceSelect* that allows selection of the type of object (class) required. An *IfcPropertyReferenceValue* may have a usage name that defines the purpose or usage

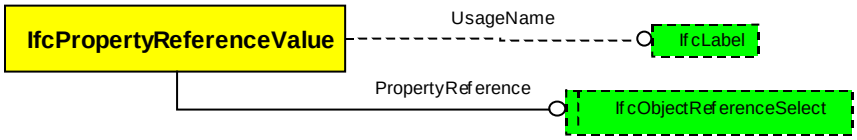


Figure 149 : Definition of IfcPropertyReferenceValue

The *IfcObjectReferenceSelect* defines the types of object (classes) that may be referenced as a property within an IfcPropertySet object. The purpose is to make available the capabilities of the class as though it was a property.

There are a limited number of classes that may be selected through the use of the *IfcObjectReferenceSelect* data type. All of these classes occur within the Resource layer of the IFC Schema. This restriction conforms to the provisions of the IFC Technical Architecture which requires that an object can only reference a class at the same or a lower layer within the Architecture. Since an *IfcPropertyReferenceValue* is itself a class that exists at the Resource layer, it can therefore only refer to other classes at the Resource layer.

10.1.6.7 Concept of Complex Property

An *IfcComplexProperty* provides the means for extending the range of properties that can be assigned to an object by defining a mechanism for bringing other properties into named groupings.

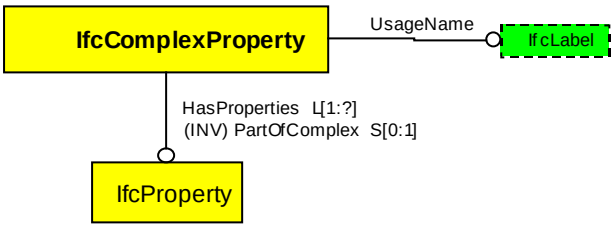


Figure 150 : Definition of IfcComplexProperty

Since any *IfcComplexProperty* can include other properties (being either *IfcSimpleProperty* or *IfcComplexProperty*) they can be assigned in a tree structure by:

- An *IfcPropertySet* contains a set of properties, one or more of which may be of type *IfcComplexProperty*.
- An *IfcComplexProperty* contains a one or more other properties amongst which may be zero, one or more of type *IfcComplexProperty*

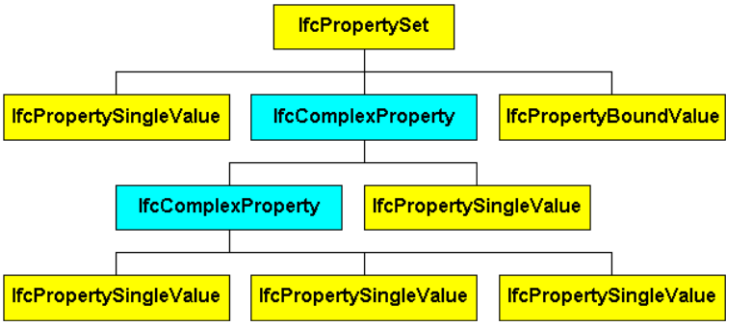


Figure 151 : Example of Nested Complex Properties

An *IfcComplexProperty* has a *UsageName* that describes how the property is to be used within a property set. This is particularly important where there may be more than one complex property of the same type used within a property set. This can happen where there is more than one item of the same type within an object but the usage of each item differs.

Example:

> *Consider two complex properties of the same name that include glazing properties. The Name attribute of the complex property could be Pcomplex_GlazingProperties. The UsageName attribute for one instance of the complex property could be OuterGlazingPlane whilst for the other instance, the UsageName attribute could be InnerGlazingPane. This would distinguish the usage of the IfcComplexProperty for the two glazing panes.*

A rule on the IfcComplexProperty class prevents the assignment of more than one copy of identical complex properties within a property set.

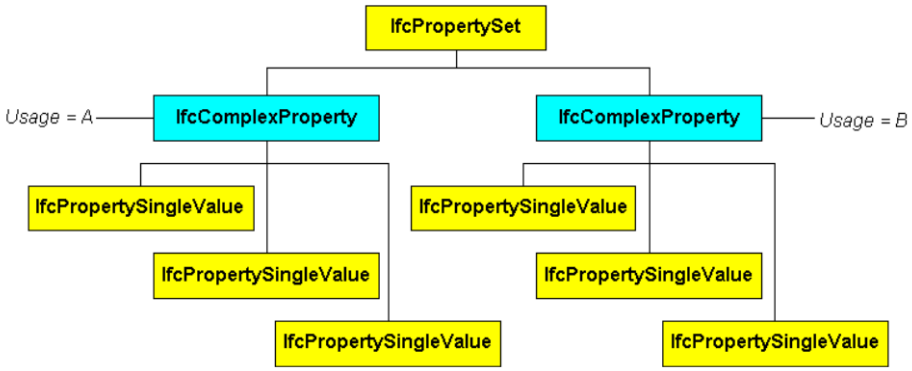


Figure 152 : Example of Complex Properties with Different Usage

10.2 Material Definitions

10.2.1 Associating material definition with objects in IFC model

Objects in IFC model that can have an associated material definition. All instances of subtypes of the *IfcElement* (with the exception of *IfcOpeningElement*) can have an instantiated inverse relationship *HasAssociations* pointing to *IfcRelAssociatesMaterial*. This provides a pointer to *IfcMaterialSelect*, where the actual instance is either

- of type *IfcMaterial* (to be used in case of one single solid material),
- of type *IfcMaterialList* (for multiple material elements when precise structure is not specified), or
- of type *IfcMaterialLayerSetUsage* (for layered elements, where the structure is specified).

In the IFC model the material association is defined as (fully attributed view):

```
ENTITY IfcRelAssociatesMaterial;
   (* ENTITY IfcRoot *)
      GlobalId :  IfcGloballyUniqueId;
      OwnerHistory :  IfcOwnerHistory;
      Name : OPTIONAL IfcLabel;
      Description : OPTIONAL IfcText;
   (* ENTITY IfcRelAssociates *)
      RelatedObjects  : SET [1:?] OF IfcRoot;
   (* ENTITY IfcRelAssociatesMaterial *)
      RelatingMaterial : IfcMaterialSelect;
END_ENTITY;

TYPE IfcMaterialSelect = (
      IfcMaterial,
      IfcMaterialList,
      IfcMaterialLayerSetUsage);
END_TYPE;
```

The material association relationship allows to assign the same material definition to multiple instances of *IfcElement*, or *IfcTypeObject*, and thereby it allows to share the same material information across many objects. Therefore the material information (such as material layer sets, material lists and materials) can be stored and exchanges separately from the actual building elements.

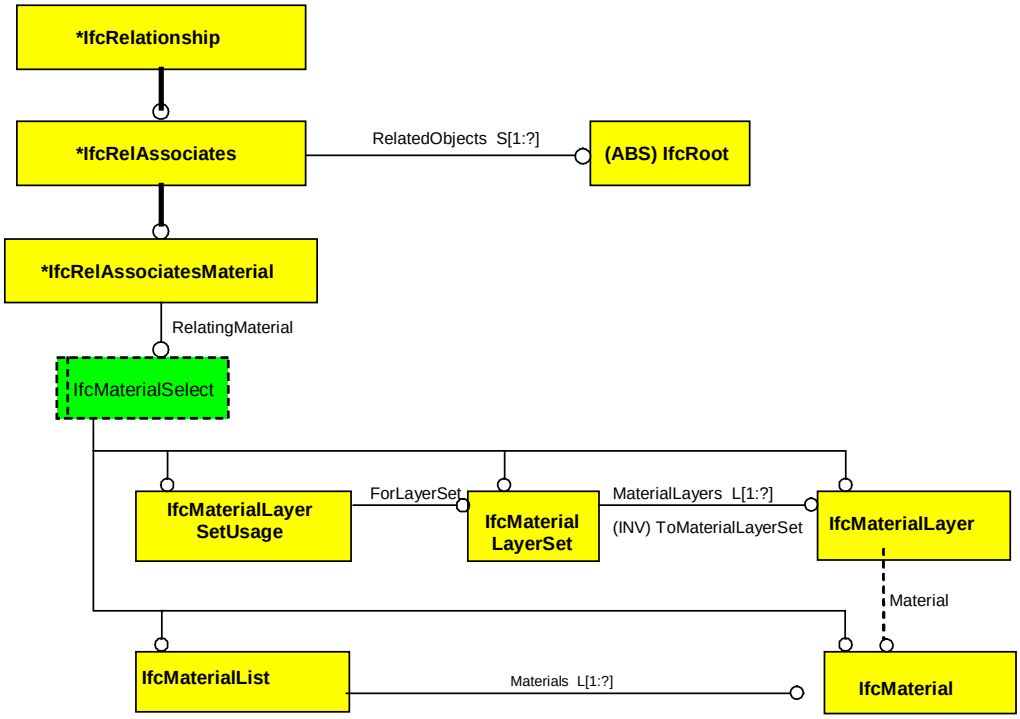


Figure 153 : Definition of material assignment

NOTE: In IFC2x2 the IfcMaterialLayerSet and the IfcMaterialLayer have been added to the IfcMaterialSelect and can now be directly assigned to IfcObject and IfcTypeObject.

10.2.1.1 Associating a single material

In the first case, where there is a solid element, e.g. a concrete wall, the select item *IfcMaterial* would be used. This assigns the same material to the complete body of the element. The result is an isotropic material definition for the element.

```
ENTITY IfcMaterial;
      Name : IfcLabel;
   INVERSE
      ClassifiedAs : SET [0:1] OF IfcMaterialClassificationRelationship FOR
                     ClassifiedMaterial;
END_ENTITY;
```

Example:

Three concrete column are exchanged by referencing the same material definition.

```
#11=IFCCOLUMN('abcdefghijklmnopqrst11',#21,$,$,$,$,$,$);
#12=IFCCOLUMN('abcdefghijklmnopqrst12',#22,$,$,$,$,$,$);
#13=IFCCOLUMN('abcdefghijklmnopqrst13',#23,$,$,$,$,$,$);
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#24,$,$,(#11,#12,#13),#4);
#4=IFCMATERIAL('Concrete');
```

10.2.1.2 Associating a list of materials

In the second case, where there is an element made of several materials, e.g. a wall of concrete, brickwork and mineral wall insulation, but the exact structural configuration is not given, the select item *IfcMaterialList* would be used:

Note: The use of material list for walls is only allowed for arbitrary walls (i.e. instances of IfcWall, which are not instances of IfcWallStandardCase).

```
ENTITY IfcMaterial;
      Materials : LIST OF [1:?] IfcMaterial;
END_ENTITY;
```

Example:

The IFC file could appear for the case of the arbitrary wall as:

```
#1=IFCWALL('abcdefghijklmnopqrst01',#21,$,$,$,$,$,$);
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#22,$,$,(#1),#4);
#4=IFCMATERIALLIST(#5,#6,#7);
#5=IFCMATERIAL('Concrete');
#6=IFCMATERIAL('Brick');
#7=IFCMATERIAL('Mineral wool');
```

10.2.1.3 Associating material layers

In the third case, where there is an element made of several materials, e.g. a layered wall made of concrete, brickwork and mineral wall insulation, and the exact structural configuration is given, the select item *IfcMaterialLayerSetUsage* should be used. This also allows to reflect the material layers within the presentation of the element:

```
ENTITY IfcMaterialLayerSetUsage;
      ForLayerSet  : IfcMaterialLayerSet;
      LayerSetDirection  :  IfcLayerSetDirectionEnum;
      DirectionSense  :  IfcDirectionSenseEnum;
      OffsetFromReferenceLine  : IfcLengthMeasure;
END_ENTITY;

TYPE IfcLayerSetDirectionEnum = ENUMERATION OF (AXIS1, AXIS2, AXIS3);
END_TYPE;

TYPE IfcDirectionSenseEnum = ENUMERATION OF (POSITIVE, NEGATIVE);
END_TYPE;
```

Here the relationship *ForLayerSet* points to the instance of *IfcMaterialLayerSet* that is used to describe the material information of this particular wall. The attributes '*LayerSetDirection*', '*DirectionSense*' and *OffsetFromReferenceLine* give the orientation and location of the layer set relative to the wall geometry:

Note, that one instance of IfcMaterialLayerSet can be used by several walls, i.e. the same material combination may be assigned with different offsets to several walls.

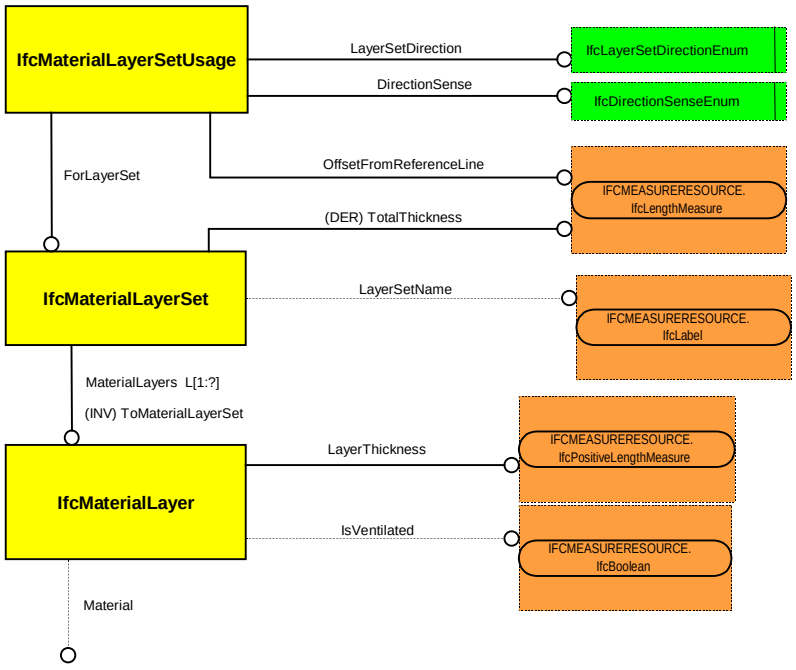


Figure 154 : Definition of material layer set

For a layered wall, the *IfcMaterialLayerSet.TotalThickness* shall be equal to the wall thickness (measured in the Y dimension of the extrusion coordinate system), and the layer set Y direction shall coincide with the extrusion profile positive or negative Y-direction (depending on the *DirectionSense*

attribute). Thus, the attribute *LayerSetDirection* shall be set to AXIS2. The reference line shall be the wall axis path.

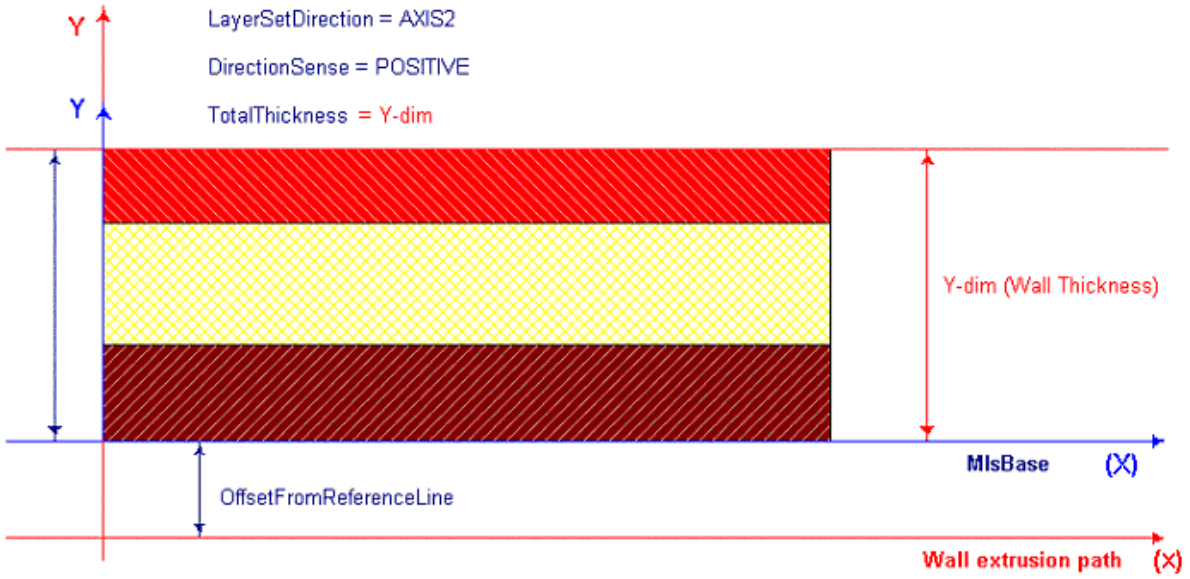


Figure 155 : Usage of material layer set

Figure 154 shows the definition of the material layer set being outside of the wall path, starting from the *OffsetFromReferenceLine* (positive offset) and assigning all layers into the same positive direction.

Example

In a case, where a wall (and its material layer set) with vertically extruded geometry has been assigned centric to the wall path, the IFC file could be as:

```
#1=IFCWALL('abcdefghijklmnopqrst01',#21,$,$,$,$,$,$);
#3=IFCRELASSOCIATESMATERIAL('abcdefghijklmnopqrst02',#22,$,$,(#1),#4);
#4=IFCMATERIALLAYERSETUSAGE(#5,.AXIS2.,.POSITIVE.,-150.);
#5=IFCMATERIALLAYERSET((#6,#7,#8),'Layered wall type 1');
#6=IFCMATERIALLAYER(#9,100.,$);
#7=IFCMATERIALLAYER(#10,120.,$);
#8=IFCMATERIALLAYER(#11,80.,$);
#9=IFCMATERIAL('Concrete');
#10=IFCMATERIAL('Mineral wool');
#11=IFCMATERIAL('Brick');
```

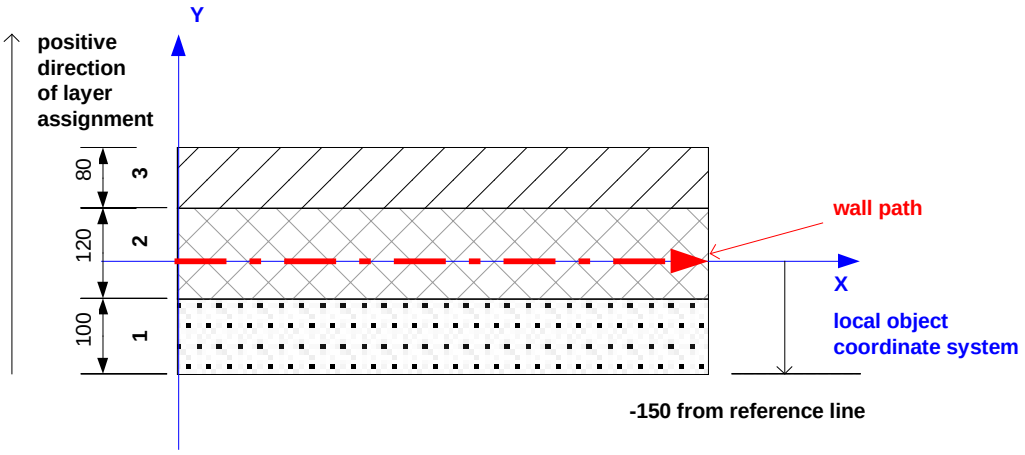


Figure 156 : Example for material layer set use assigned to a wall

10.2.2 Material Classifications

Classifications of a material can be expressed using general *IfcClassificationReference* (see the discussion about the concept of classification in 10.4.2), associated with specific material through *IfcMaterialClassificationRelationship*, defined as:

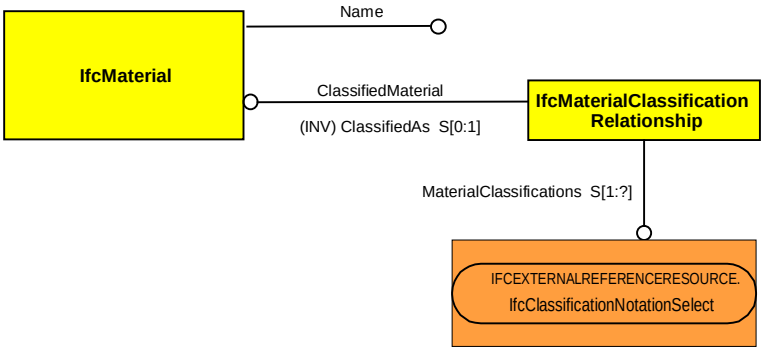


Figure 157 : Definition of Material Classification

Example:

In case of a particular steel material "S275JR" (formerly "Fe430B") according to Euronorm EN10025 (1993), this could appear in the exchange file as:

```
#20=IFCMATERIAL('Steel');
#30=IFCMATERIALCLASSIFICATIONRELATIONSHIP((#40),#20);
#40=IFCCLASSIFICATIONREFERENCE ($,'S275JR','Fe430B',#41);
#41=IFCCLASSIFICATION('CEN','1',$,'EN10025');
```

10.3 Quantities and Measurement of Elements

'Methods of measurement' are frequently used as the basis on which elements may be measured for inclusion in cost or progress submissions. They may include sets of rules that determine how the measurement should be established and the units of measure in which it should be expressed. The rules may be developed from some abstract concepts of measurement rather than actual values (such as length, area etc.) that might otherwise be physically associated with an object or that might be derived through knowledge of the geometric representation of an object.

Example:

Consider a straight section of pipework with a 90° bend at one end and a square tee at the other end.

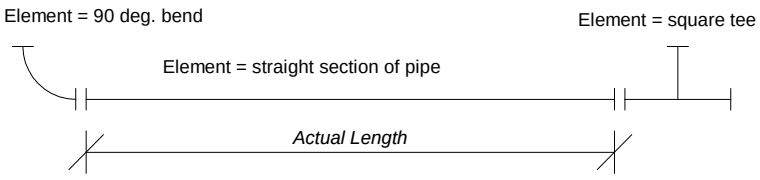


Figure 158 : Determining actual length

The actual length of the pipe section (as cut and fitted) is measured as the distance between each physical end and should include for the actual length of the pipe that is inserted into the fitting in the case of pipework with screwed or capillary joints. In geometric terms, it is normally measured as shown; that is the length of piping between the end of the bend to which it is connected and the end of the tee to which it is connected.

The actual fitting elements can also be determined as being a tee and a bend.

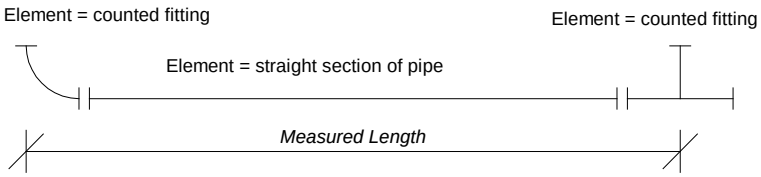


Figure 159 : Determining measured length

A method of measurement might however set a rule that the length should be measured as the distance between the ends of the section of pipe if they were projected to a point at which they would meet with an adjacent section of pipe. This would mean that the measured length would be greater than the actual length but would be considered satisfactory under the rules of the method of measurement for costing.

A further rule might be that instead of determining the type of each fitting, all fittings would be treated as being the same (called a 'counted fitting' in this example). With such a rule, manual costing is easier to achieve. A further alternative rule for determining the length of pipework is to make an 'extra-over' allowance for fittings that is added to the length of the pipework. This might take the form of a value or percentage. In this case, costs would be determined only on the measured length and unit cost of the pipework, the fittings not being specifically costed at all.

The quantities and methods of measurements are used to fulfill the UoF "Element Quantities", which are utilized at *IfcSpatialStructureElement* and *IfcElement*. They are required to enable the exchange scenario of "quantity takeoff" and will certainly be included in an IFC2x view definition generated to provide for quantity takeoff using IFC. The generic definitions, as introduced within this section, provide for the flexibility needed to meet the particular requirements of local or regional methods of measurements.

10.3.1 Concept of Element Quantity

Quantities can be assigned to objects through the *IfcElementQuantity* class. This is defined in the *IfcProductExtension* schema. It is one of the subtypes of *IfcPropertySetDefinition* that is explicitly defined as part of the IFC data definition in the EXPRESS language which means that all of the properties within the *IfcElementQuantity* property set have defined semantics[20].

Each instance of *IfcElementQuantity* can contain one or more instances of *IfcPhysicalQuantity* (q.v.). This means that all defined quantity properties that are to be related to an object can be collected together into a single instance of *IfcElementQuantity*. It also means that a single instance of *IfcElementQuantity* can be related to many objects thereby minimizing the data load that has to be carried by individual objects.

Each instance of *IfcElementQuantity* also has a single defined *MethodOfMeasurement* attribute. This is a string value that names the rule set that is used for definition of the quantity properties. All quantities contained by a single instance of *IfcElementQuantity* must conform to the same method of measurement. However, should it be necessary to use quantities defined by more than one method of measurement; it is possible to make more than one relationship of *IfcElementQuantity* to an object.

NoteThis means, that for example the "running meter" and the "side area" of a wall, if measured according to the same method, like the German VOB, would be attached to the wall instance by a single instance of IfcElementQuantity. If additionally the volume of the wall is derived by a different method, like the German DIN276, that it is attached by a separate instance of IfcElementQuantity.

[20] This is as opposed to properties within a property set that is defined through use of the IfcPropertyResource meta model in which each property only has the semantics of a name the semantics of which must be declared or agreed particularly.

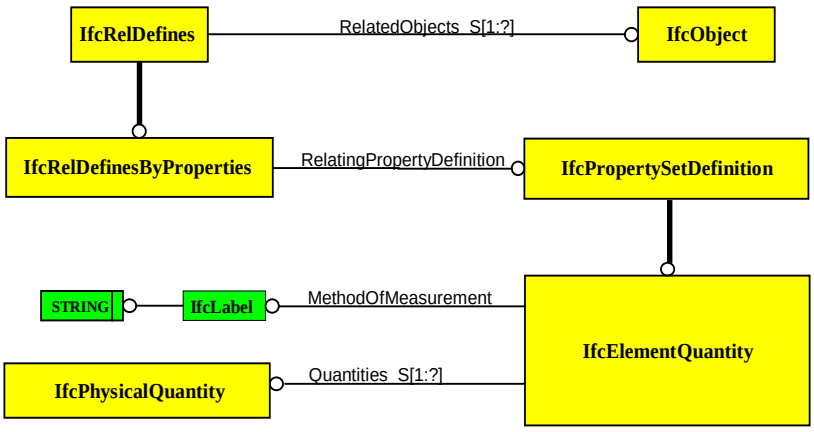


Figure 160 : Definition of Element Quantity

10.3.1.1 Concept of Associating Quantities to Objects

Quantities can be defined for any instance of a subtype of *IfcObject*. This means that, as well as individual physical products, quantities can also be defined for processes, resources, groups, controls etc.

Quantities are related to objects through the *IfcRelDefinesByProperties* class in the *IfcKernel* schema. This relationship class can apply a single *RelatingPropertyDefinition* to one or many instances of a subtype of *IfcObject*. It should be noted that it is possible to relate a single *IfcElementQuantity* to different types of *IfcObject*; there is no limiting rule that says that one *IfcElementQuantity* can only be related to one type of *IfcObject*.

The attribute *MethodOfMeasurement* defines the method which has been used to create the quantities of the elements. These methods are usually national standards that define the measurement rules. Examples are VOB for Germany or UMM3 for UK, for general quantities that are not subjected to national rules the following values for *MethodOfMeasurement* are to be used:

- "" (empty string) – no specification given
- "physical" – physically correct values from the geometric representation used

Example:

> *Consider two standby generators for providing electrical supply in the event of a power failure. For the purposes of this example, both of these are considered to be of the same type (IfcElectricGeneratorType) but having a different function. In this case, the quantity of interest is the weight according the third edition of the Unidentified Method of Measurement (known as UMM3). The value that is assigned to the units is 500kg.This is taken to include for a full fuel tank at the generator.*

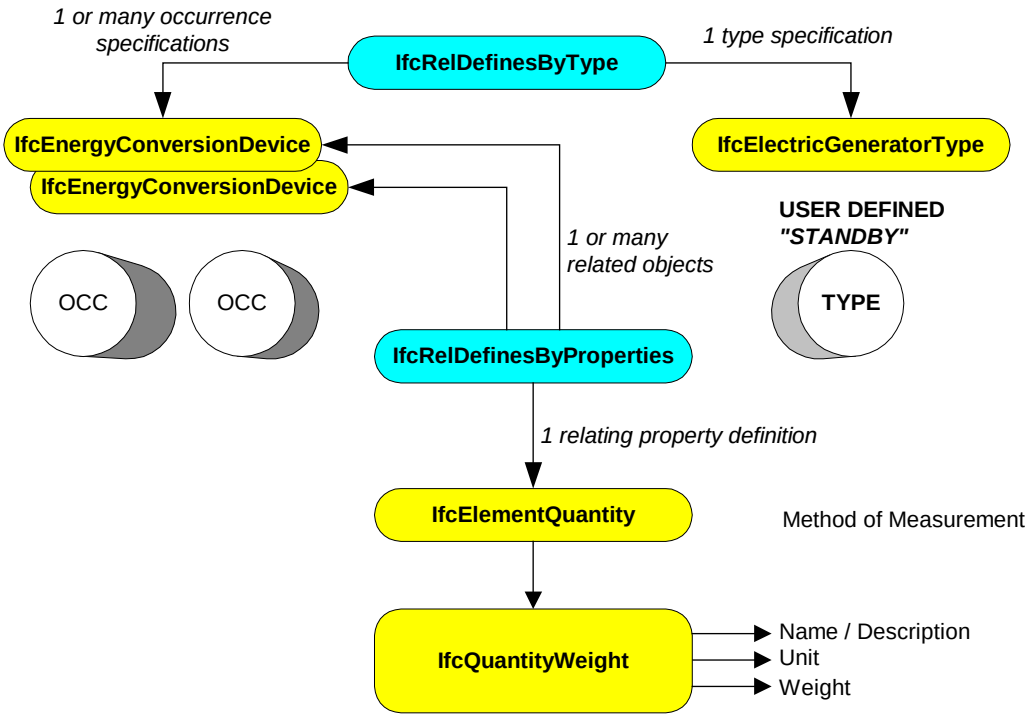


Figure 161 : Usage of IfcElementQuantity

***IFC2x2 usage**: Information concerning the type of generator is specified using IfcElectricGeneratorType. The fact that it is a standby generator is specified through the enumeration IfcElectricGeneratorTypeEnum. Practically, this enumeration only specifies UserDefined and NotDefined types. Therefore, for the purposes of this example, a user defined type will be selected and a user defined value of 'STANDBY' assigned. This is achieved using the IfcElectricGeneratorType.ElementType attribute (inherited through IfcElementType).*

An occurrence of a generator is specified through IfcEnergyConversionDevice.

```
/* general definitions for ownership */
#100=IFCOWNERHISTORY(#106, #107, .READWRITE., .NOCHANGE., $, $, $, 978921854);
#101=IFCOWNERHISTORY(#106, #107, .READWRITE., .NOCHANGE., $, $, $, 978921854);
#102=IFCOWNERHISTORY(#106, #107, .READWRITE., .NOCHANGE., $, $, $, 978921854);
#106=IFCPERSONANDORGANIZATION(#108, #109, $);
#107=IFCAPPLICATION(#109, '1', 'IFC2x Text Program', 'IFC2x Test');
#108=IFCPERSON($, 'Miller', 'Frank', $, $, $, $, $);
#109=IFCORGANIZATION($, 'IAI', 'International Alliance for Interoperability', $, $);

/* definition of element quantity containing a single physical quantity */
/* as the unit is not given it refers to the global unit assignment *
#400=IFCQUANTITYWEIGHT ('Generator Weight','Includes for fuel provision',$,500.0);
#600=IFCELEMENTQUANTITY ('gabcdeghijklmnopqrst97',#101,$,$,'UMM3',(#400));

/* assigning quantities to occurrences */
#700=IFCRELDEFINESBYPROPERTIES ('gabcdeghijklmnopqrst700',#102,$,$,(#950,#960),#600);

/* specification of type */
#800=IFCELECTRICGENERATORTYPE ('gabcdeghijklmnopqrst800',#100,$,$,$,$,$,$,
     'STANDBY',.USERDEFINED.);

/* relationship between type and occurrence */
#900=IFCRELDEFINESBYTYPE('abcdefghijklmnopqrst900', #2, $, $, (#950,#960), #800);

/* elements to which the same quantity is assigned (or shared) */
#950=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst950',#100,$,$,$,#10001,#10002,$);
#960=IFCENERGYCONVERSIONDEVICE('abcdefghijklmnopqrst960',#100,$,$,$,#10003,#10004,$);

/* #10001 - #10004 inc.give object placement and representation of occurrences */
```

10.3.1.2 Concept of Physical Quantity

The *IfcPhysicalQuantity* defines the various forms of quantity properties that can be contained within the *IfcElementQuantity*. Five types of quantities are defined:

- *Length* (*as IfcQuantityLength*)
- *Area* (*as IfcQuantityArea*)
- *Volume* (*as IfcQuantityVolume*)
- *Count* (*as IfcQuantityCount*)
- *Weight* (*as IfcQuantityWeight*)

Each type of *IfcPhysicalQuantity* has a value attribute that is defined according to the equivalent defined data type within the *IfcMeasureResource* schema e.g. the value of an *IfcQuantityLength* is given by an *IfcLengthMeasure*.

Each instance of *IfcPhysicalQuantity* must have a name that defines the context for its usage. For instance, a wall may have several area measures. These could have context names such as footprint area, left wall face area, right wall face area etc. The areas would be given by three instances of the area quantity subtype, with different Name string values. The Name attribute defines the actual usage or kind of measure. The interpretation of the name label has to be established within the actual exchange context.

Additionally, each quantity may have an associated informative description that can be used to explain or further qualify the Name attribute. Each instance of *IfcPhysicalQuantity* may also have a unit. The definition of units is introduced in section 3.3.

If the unit is given, the value for unit overrides the global setting of the project-wide units within *IfcProject.UnitsInContext*. If the value for unit is omitted, than the unit defined within *UnitsInContext* is used. In order to find the applicable unit, a measure type is provided for each measure value.

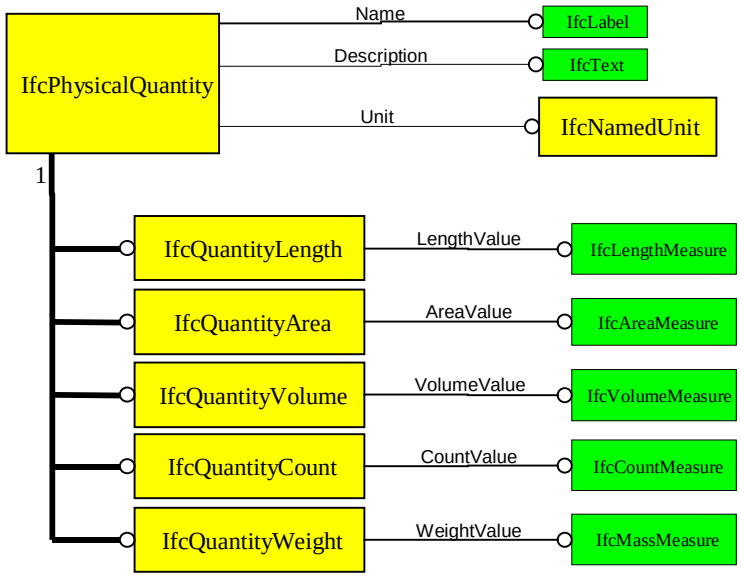


Figure 162 : Definition of Physical Quantity

The five subtypes reflect the major measure types used in the various methods of measurement around the world.

10.4 Referencing External Information

The IFC Model allows to reference information, which is externally defined. In this case only the access information is stored within the IFC exchange set (mostly the URL for HTTP access protocol), and eventually some header information (e.g. the name of the reference, a short summary, etc.). The actual content however remains at the source location.

There are three types of external references within the IFC2x model:

- document references
- classification references
- library references

10.4.1 Referencing External Documents

to be added after version 1.0 of the document

10.4.2 Referencing External Classifications

It is recognized that there are many different classification systems in use throughout the AEC/FM industry and that their use differs according to geographical location, industry discipline and other factors. For a generic model such as IFC, it is necessary to allow for the adoption of any rational classification system whether it is based on elements, work sections or any other classifiable division.

The classification model is able to represent classifications according to the most advanced current concepts from work in ISO TC59, ICIS (International Construction Information Society) and EPIC (European Product Information Coding) as well as more traditional classifications such as those in the various SfB forms used internationally, CAWS, Master format etc., or other national classification system, like the DIN in Germany.

The classification model forms part of the *IfcExternalReferenceResource* schema and specifies the use of the independent resources necessary for the scope and information requirements for the exchange and sharing of classification information between application systems. Such information may be used at all stages of the life cycle of a building.

The following are within the scope of the classification model in IFC2x:

- The provision of one or more classification notations to an object.
- The inclusion of one or more facets to a classification notation.
- Referencing of facets of a classification notation from a described source (classification item or classification table)
- Exposure of the hierarchy of a classification structure.
- Identification of the source of the classification.
- The designation of a classification in terms of its source, edition and name.
- The provision of a means of semantically identifying the meaning of a classification notation.
- Referencing a classification held on an external source.

The following is out of scope of the classification model in IFC2x:

- The ability to translate from one classification notation to another.

10.4.2.1 Concept of Associating Classification to Objects

Objects are classified by attaching a classification (either light weight classification reference or a fully defined facet within a classification table) by means of a relationship instance. This relationship class, *IfcRelAssociatesClassification*, forms part of the *IfcKernel* schema. It is used to apply an *IfcClassificationReference* or *IfcClassificationNotation* to an object or an object type and property set through its relation to *IfcRoot* (from which all objects that can be classified are subtyped).

Each instance of *IfcRelAssociatesClassification* enables the association of one classification notation with one or many objects. See also section 10.4.2.7 for the concept of association of classifications.

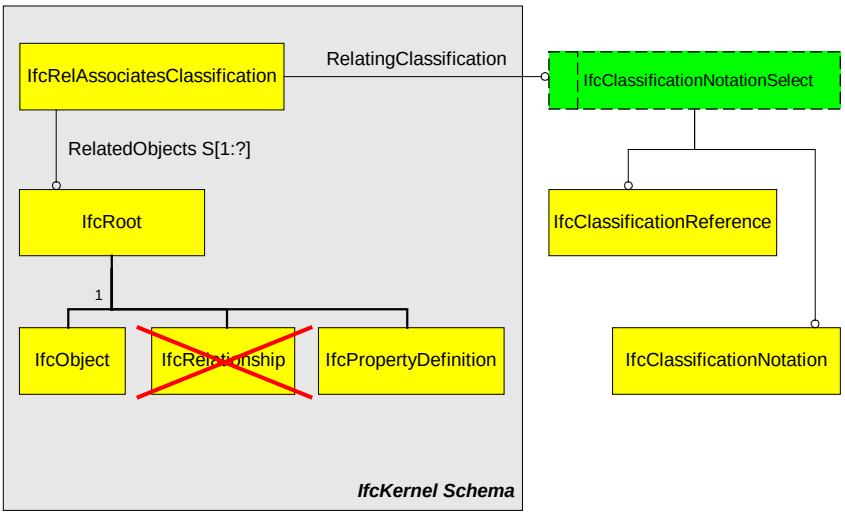


Figure 163 : Definition of associating a classifcation

NOTE: By virtue of a where rule at the IfcRelAssociates, relationships are prohibited from having external references or definitions.

It is possible for an object to have multiple classifications. This is achieved through the use of multiple instances of *IfcRelAssociatesClassification*, each instance pointing to a particular classification notation and to the objects that are classified by this notation.

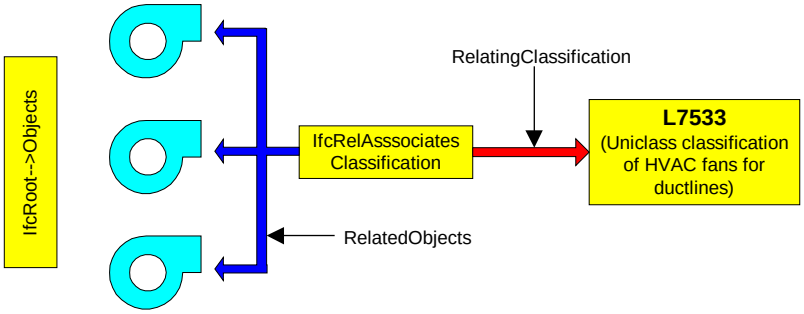


Figure 164 : Example of attaching a classification to multiple instances

Using the *IfcRelAssociatesClassification* relationship class means that any object that is not classified does not have to carry empty attributes for classification (as was the case with previous versions of the IFC model). This also facilitates a better subdivision of the IFC2x model into implementation views.

10.4.2.2 Concept of Classification Notation

The principal applied is that any type of object can be classified. No distinction is made between a product, a process, a control or a resource. The means by which an object may be classified is the *IfcClassificationNotation* class. An *IfcClassificationNotation* is a notation used from published reference (which may be either publicly available from a classification society or is published locally for the purposes of an organization, project or other purpose).

Note that a classification notation may be developed using various notation facets. A facet is a part of the actual notation but which has a specific meaning. For instance, it may be appropriate to classify an item by owning actor (represented by A=Architect) and by an entry from a classification table such as CI/SfB (represented by 210 for external wall). This gives a classification as A210.

All classifications of an object that are contained within the IFC model are made through the *IfcClassificationNotation* class. For a given object, the *IfcRelAssociatesClassification* class makes the connection between the *IfcObject* (abstract superclass, here: *IfcDoor* has been taken as an example for instantiation) and the *IfcClassificationNotation* (used for full classification).

IfcObject <-- *IfcRelAssociatesClassification* --> *IfcClassificationNotation*.

Example:

Consider an object that is to be classified with the notation "L6814". In this case, the IfcRelAssociatesClassification has the form:

```
#100=IFCDOOR ('gabcdeghijklmnopqrst02',#1000,.....);
#200=IFCCLASSIFICATIONNOTATIONFACET ('L6814');
#300=IFCCLASSIFICATIONNOTATION ((#200));
#400=IFCRELASSOCIATESCLASSIFICATION ('gabcdeghijklmnopqrst02',#1001,$,$,(#300),#100);
```

If the object is to have other notations (e.g. B2186 and Z6793), i.e. there are multiple classifications available to an object, then the IfcRelAssociatesClassification has the form:

```
#100=IFCDOOR ('gabcdeghijklmnopqrst02',#1000,.....);
#200=IFCCLASSIFICATIONNOTATIONFACET ('L6814');
#210=IFCCLASSIFICATIONNOTATIONFACET ('B2186');
#220=IFCCLASSIFICATIONNOTATIONFACET ('Z6793');
#300=IFCCLASSIFICATIONNOTATION ((#200,#210,#220));
#400=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#300),#100);
#400=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#300);
```

It is a requirement that a classification notation can only bring together facets from the same classification system or source. Bringing together notation facets from different sources within the same classification notation is not allowed. However, since the *IfcRelAssociatesClassification* provides for an N to M relation between classification notation and classified objects, it does allow for multiple classifications to a single object. In order to assign multiple classification notations coming from different classification systems an instance of *IfcClassificationNotation* shall be created to contain all facets from a single source, leading to multiple instances of *IfcClassificationNotation* and also to multiple instances of *IfcRelAssociatesClassification*, which assigns each notation to the same (group of) objects.

Example:

Consider an object that is to be classified with the notations from two distinct classification systems. In this case, the IfcRelAssociatesClassification has the form:

```
#100=IFCDOOR ('gabcdeghijklmnopqrst02',#1000,.....);
#200=IFCCLASSIFICATIONNOTATIONFACET ('L6814');
#210=IFCCLASSIFICATIONNOTATIONFACET ('B2186');
#220=IFCCLASSIFICATIONNOTATIONFACET ('Z6793');
#300=IFCCLASSIFICATIONNOTATIONFACET ('106');
#400=IFCCLASSIFICATIONNOTATIONFACET ('A-31-623');
#410=IFCCLASSIFICATIONNOTATIONFACET ('B-62-562');
#500=IFCCLASSIFICATIONNOTATION ((#200,#210,#220));
#510=IFCCLASSIFICATIONNOTATION ((#300));
#520=IFCCLASSIFICATIONNOTATION ((#400,#410));
#600=IFCRELASSOCIATESCLASSIFICATION ('gabcdeghijklmnopqrst02',#1001,$,$,
    (#500,#510,#520),#100);
#610=IFCRELASSOCIATESCLASSIFICATION ('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#510);
#620=IFCRELASSOCIATESCLASSIFICATION ('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#520);
#630=IFCRELASSOCIATESCLASSIFICATION ('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#530);
```

10.4.2.3 Concept of Classification Notation Facet

Many modern classification systems (such as Uniclass, BSAB etc.) allow for multi-faceted classification. This is reflected in the classification model through provision of the *IfcClassificationNotationFacet* class where each instance represents one facet of a classification notation. The classification notation itself is then made up of a list of notation facets. The list is declared to be a unique list to ensure both that there is order in the specification of the notation facets and to ensure that a facet can only be included once in each list.

An *IfcClassificationNotationFacet* object holds an individual classification value that is to be assigned to an object through *IfcClassificationNotation* and *IfcRelAssociatesClassification* objects. An *IfcClassificationNotationFacet* is a group of alphanumeric characters used within a classification notation.

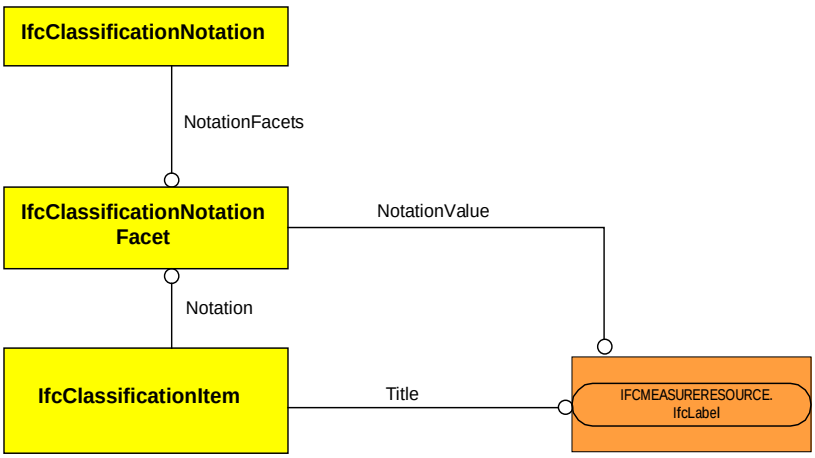


Figure 165 : Definition of a classification facets and items

Example:

> *For instance, considering a fan in an HVAC ductline. As a construction product, it has the Uniclass notation facet L7533. However, if we consider a work section view with the fan as part of the low velocity air conditioning system, the notation facet is JU30. Therefore, there are two facets in the classification notation which becomes ('JU30', 'L7533')*

10.4.2.4 Concept of Classification Item

An *IfcClassificationItem* is a class of classification notations used. Note that the term 'classification item' is used in preference to the more usual (but deprecated) term 'table' for improved flexibility. For example, the classification item "L681" in Uniclass may be used to contain all subsequent notation facets within that class of classifications which has the title "Proofings, insulation"(e.g. L6811, L6812, L6813 etc.).

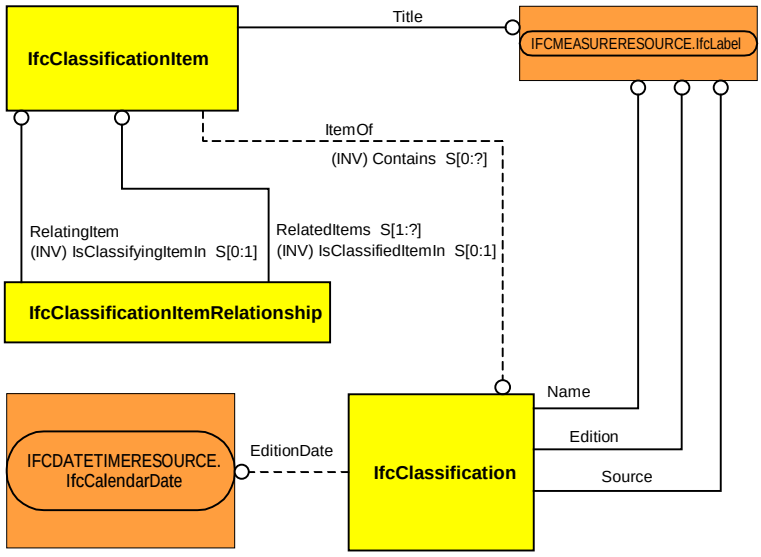


Figure 166 : Definition of classification system

10.4.2.5 Concept of Classification Item Relationship

Classification systems are generally declared in a hierarchical structure in which a facet at an upper level in the hierarchy (parent level) is a generalization of the facets at the next lower level in the hierarchy (child level). The hierarchy of the classification system is defined in tables (ref. CI/SfB) or sections (ref. CAWS) or some other named, coherent approach. Typically, the position in the hierarchy is identified by a nomenclature or label e.g. X12, and the identity at the child level is derived by adding (concatenating) characters e.g. X121, X122, X123 etc.

Within the classification model, the ability to identify location in a classification hierarchy is termed the *IfcClassificationItem* (so as not to be identified as table, section etc. but as a generalizations of these terms).

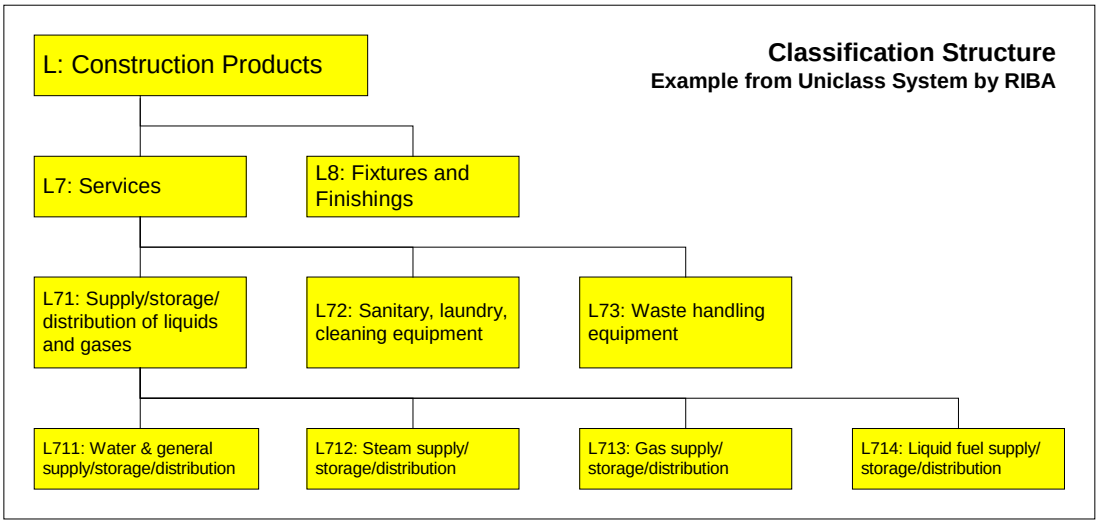


Figure 167 : Example of a hierarchical classification system

The whole of the classification hierarchy can now be exposed through the IFC Model. This is achieved by providing the recursive relationship class *IfcClassificationItemRelationship*. As a further restriction; a classification hierarchy cannot contain any instance of *IfcClassificationItem* more than once.

An *IfcClassificationItemRelationship* is a relationship class that enables the hierarchical structure of a classification system to be exposed through its ability to contain related classification items and to be contained by a relating classification item. *IfcClassificationItem*'s can be progressively decomposed using the *IfcClassificationItemRelationship* such that the relationship always captures the information about the parent level (relating) item and the child level (related) items of which there can be many. The following example shows how this could be achieved for the Uniclass system.

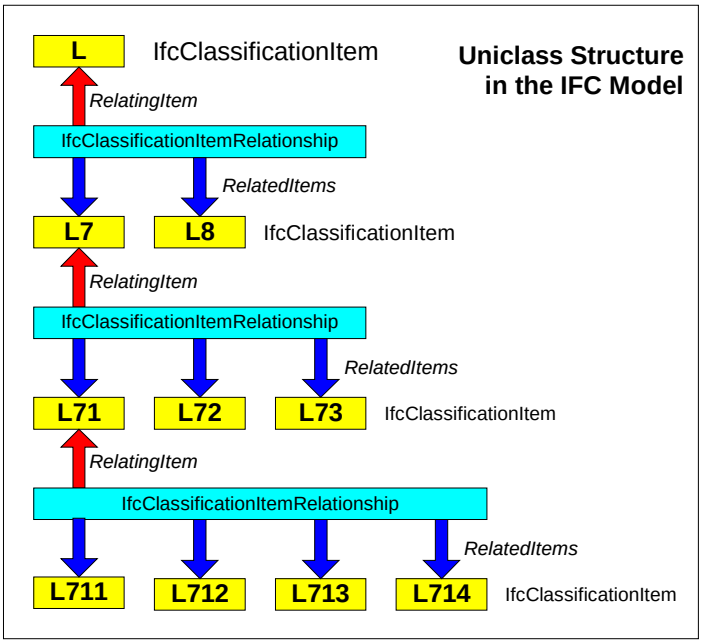


Figure 168 : Example of building a classification system in IFC2x

The inverse relationships from *IfcClassificationItem* to *IfcClassificationItemRelationship* enable information about the relationship to be recovered by the items concerned so that they are also aware of the decomposition. The cardinality of the inverse relationship is that an *IfcClassificationItem* can be the classifying item in maximum one relationship and can be a classified item in maximum one relationship. This reflects typical classification approaches that use strict hierarchical decomposition (or taxonomy) and do not have matrix relationships.

Example:

The example used in Figure 167 shows a decomposition detail from the UNICLASS system. This can be seen from:

```
#200=IFCCLASSIFICATIONITEM (#100,#2,'Door Panel');
#201=IFCCLASSIFICATIONITEM (#101,#2,'Door Panel');
#202=IFCCLASSIFICATIONITEM (#102,#2,'Door Panel');
#203=IFCCLASSIFICATIONITEM (#103,#2,'Door Panel');
#204=IFCCLASSIFICATIONITEM (#104,#2,'Door');
#205=IFCCLASSIFICATIONITEMRELATIONSHIP (#204,(#200,#201,#202,#203));
```

10.4.2.6 Concept of Classification System

Each classification item belongs to an *IfcClassification*. This provides the means to identify the classification system being used. *IfcClassification* has attributes that define its source, edition and name.

- *Name* - identifies what would usually be considered to be the name of the classification system such as CI/SfB, BSAB, CAWS, Masterformat, Uniformat etc.
- *Edition* - provides a version identification.
- *EditionDate* - identifies the date at which the edition became operational.
- *Source* - identifies the publishing reference of the classification system (e.g. for Uniclass, the source would be RIBA)

An *IfcClassification* is used for the arrangement of objects into a class or category according to a common purpose or their possession of common characteristics.

Example:

The objective is to minimize the number of IfcClassification objects contained within a populated IFC model. Ideally, each classification system or source used should have only one IfcClassification object. However, because multiple classification is allowed, there may be many IfcClassification objects used, each identifying a different classification system or source. An example of the use of the IfcClassification class, defining a single classification item is:

```
#1=IFCCALENDARDATE (31,8,1997);
#2=IFCCLASSIFICATION('RIBA','1',#1,'Uniclass');
#3=IFCCLASSIFICATIONITEM (#104,#2,'Door');
```

10.4.2.7 Concept of Classification Referencing

Recognizing that it may not be appropriate to maintain the whole detail of classification within the schema, and taking the view that information may well be referenced from external sources by its address to enable access through mechanisms such as the World Wide Web, the classification model of IFC2x includes a means to reference a classification through the *IfcClassificationReference* class. This is a subtype of *IfcExternalReference* that has a label (which can be the reference address) and identifier. Additional information may be available concerning the classification system source that is being referenced.

An *IfcClassificationReference* is a reference into a classification system or source (see *IfcClassification*). An optional inherited "*ItemReferenced*" key is also provided to allow more specific references to classification items (or tables) by type.

10.4.2.7.1 Lightweight Classification

The *IfcClassificationReference* can be used as a form of 'lightweight' classification through the *'ItemReference'* attribute inherited from the abstract *IfcExternalReference* class. In this case, the *'ItemReference'* could take (for instance) the Uniclass notation "L6814" which, if the classification was well understood by all parties and was known to be taken from a particular classification source, would be sufficient. This would remove the need for the overhead of the more complete classification structure of the model.

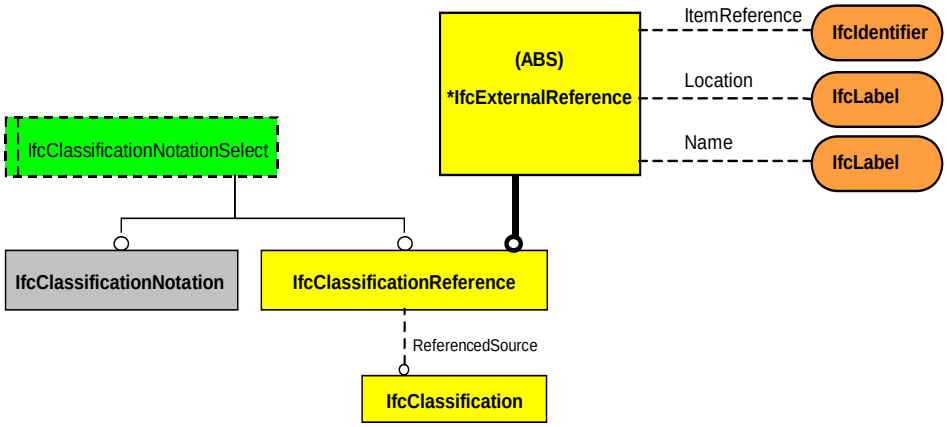


Figure 169 : Definition of a classification reference

However, it is not recommended that this lightweight method be used in cases where more than one classification system is in use or where there may be uncertainty as to the origin or meaning of the classification.

10.4.2.7.2 Referencing from an External Source

Classifications of an object may be referenced from an external source rather than being contained within the IFC model. This is done through the *IfcClassificationReference* class.

Example:

> *Consider the Uniclass notation "L6814" which has the title "Tanking". In this case, the optional attribute 'ItemReference' uses the notation "L6814", and the attribute "Name" uses the title 'Tanking' that would otherwise be applied to the IfcClassificationItem (if it was to be contained in the model).*
>
> *The location of the classification reference may be found from a classification server that is available via the Internet, like http://www.ncl.ac.uk/classification/uniclass/tanking.htm#6814.*

```
#1=IFCCALENDARDATE(31,8,1997);
#2=IFCCLASSIFICATION(‘RIBA’,’1’,#1,’Uniclass’);

#100=IFCDOOR('gabcdeghijklmnopqrst01',#1001, ..... );
#300=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#200),#100);
#300=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#200);

#200=IFCCLASSIFICATIONREFERENCE
('http://www.ncl.ac.uk/classification/uniclass/tanking.htm#6814','L6814','Tanking',#2);
```

Because the relation between *IfcRelAssociatesClassification* and classification is actually made at the *IfcClassificationNotationSelect* class that allows classification to be either contained (full internal classification) or referenced (lightweight classification), it is possible to assign both contained and referenced classifications to an object by virtue of having multiple instances of *IfcRelAssociatesClassification*.

```
#1=IFCCALENDARDATE (31,8,1997);
#2=IFCCLASSIFICATION (‘RIBA’,’1’,#1,’Uniclass’);

#100=IFCDOOR ('gabcdeghijklmnopqrst01',#1001,.....);
#500=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(300,#400),#100);
#510=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#300);
#520=IFCRELASSOCIATESCLASSIFICATION('gabcdeghijklmnopqrst02',#1001,$,$,(#100),#400);

#200=IFCCLASSIFICATIONNOTATIONFACET ('L6814');
#210=IFCCLASSIFICATIONNOTATIONFACET ('B2186');
#220=IFCCLASSIFICATIONNOTATIONFACET ('Z6793');
#300=IFCCLASSIFICATIONNOTATION ((#200,#210,#220));
#400=IFCCLASSIFICATIONREFERENCE
('http://www.ncl.ac.uk/classification/uniclass/tanking.htm#6814','L6814','Tanking',#2);
```

10.4.2.8 Translation Between Classification Notations

Whilst several different classification notations may be applied to an object, there is no equivalence between the classification notations USED. Therefore, the IFC Classification Model cannot be used to translate from one classification notation to another. The reason for this is that, generally, it is possible to select from any of several different notations within a classification system for an object. The actual selection is the responsibility of the user according to circumstances. Therefore, there is a many to many relationship between classification systems for which there is no resolution at this stage of development.

10.4.3 Referencing External Libraries

It is anticipated that many property sets defined externally to the IFC Model will be referenced from a library or database of product data held by a manufacturer/supplier, an information provider acting on their behalf or some other body holding significant information sources.

Referencing information from a library allows for the data to be retained in the library rather than in the IFC Model. In an information exchange/sharing scenario, this can be useful since it means that the amount of information needing to be transferred can be minimized. The IFC Model provides a structure[21] that enables property sets either to be referenced from the library in which they are held or delivered from the library into the IFC model as a property definition that can be associated with one or more objects.

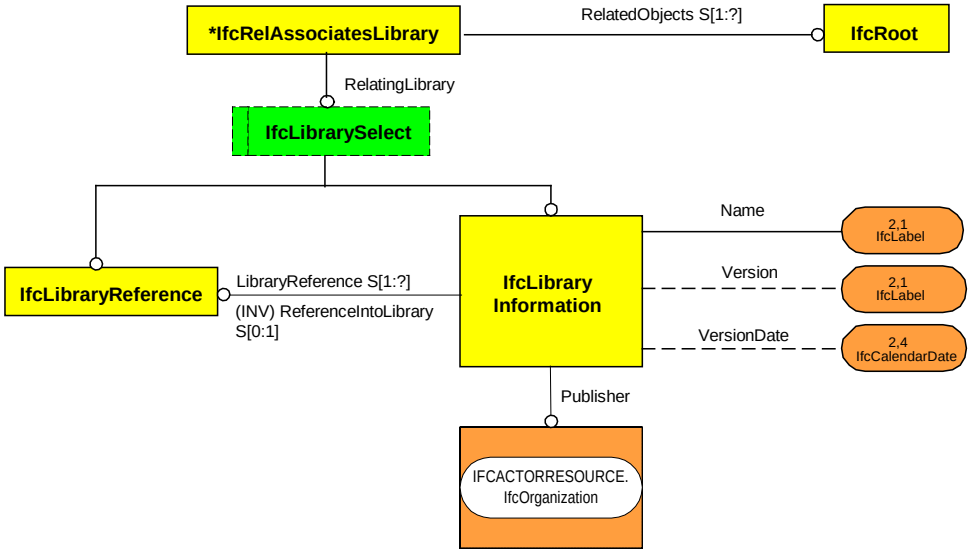


Figure 170 : The mechanism to reference libraries

The relationship class *IfcRelAssociatesLibrary* manages the linkage between the library and an object. It should be noted that the related objects in this case are instances of the IfcRoot class. This is because a selection of whether to associate the library with an *IfcObject* or an *IfcPropertyDefinition* has to be made. In this case, this cannot be done through the use of a SELECT data type since the EXPRESS language does not allow the declaration of inverse relationships from a SELECT type. Therefore, the association has to be made to the common supertype of *IfcObject* and *IfcPropertyDefinition* which is *IfcRoot*. Since *IfcRoot* also has other subtypes, a rule is applied to the relationship class which constrains the subtypes of *IfcRoot* that can have a library associated.

In selecting the *IfcLibraryReference* to be used, the attributes of the *IfcExternalReferencResource* class are inherited. These enable the location of the library to be captured. Location can be any fully qualified address such as a directory path on a CD. However, it is anticipated that it will more usually be a URL enabling referencing to occur across the World Wide Web.

An *ItemReference* may also be captured that enables a pointer into the library source to be captured. This provides a more complete identity for the information within the library.

[21] Although this discussion relates to the referencing of property definitions, the model structure discussed can also manage the association of object information from libraries.