

# Technical Roadmap

April  
**2020**



“Getting ready for the future”

*buildingSMART needs to create scalable interoperability for data standards, tools and the underlying technologies*

# Industry Foundation Classes

## IFC 4.3 developments

Now maintained in UML class diagram;  
published on GitHub

Developments on updating building domain  
and backlog of technical issues are being  
processed.  
ISO process has started.



### Quality management

Online quality management and  
transparent development tooling  
(Travis)



### Thorough testing

High vendor engagement made it  
the best tested release of IFC.



### Micro cosmos

Direct integration with bSDD,  
Translations, documentation, etc.

# Towards IFC 5 “the future of IFCs”

About 120 years of IFC experience has come together to define the principles of IFC 5.  
Nine months of fortnightly meetings sets out a first step. Still a long way to go...

## Future of the Industry Foundation Classes: towards IFC 5

Léon van Berlo, [leon.vanberlo@buildingsmart.org](mailto:leon.vanberlo@buildingsmart.org)  
buildingSMART International, London, United Kingdom

Thomas Krijnen, [t.krijnen@tudelft.nl](mailto:t.krijnen@tudelft.nl)  
Delft University of Technology, Delft, The Netherlands

Helga Tauscher, [helga.tauscher@htw-dresden.de](mailto:helga.tauscher@htw-dresden.de)  
HTW Dresden - University of Applied Sciences, Dresden, Germany

Thomas Liebich, [tl@aec3.de](mailto:tl@aec3.de)  
AEC3, Munich, Germany

Arie van Kranenburg, [management@arkey.nl](mailto:management@arkey.nl)  
Arkey Systems, Houten, The Netherlands

Pasi Paasiala, [Pasi.Paasiala@solibri.com](mailto:Pasi.Paasiala@solibri.com)  
Solibri, Helsinki, Finland

### Abstract

The buildingSMART Technical Roadmap, published in April 2020, was the start of multiple modernization efforts for buildingSMART Solutions and Standards. The modernization, modularization, and normalization of the Industry Foundation Classes (IFC) is one of the priorities. A taskforce has been working on restructuring the core of IFC for the IFC 5 developments. The following topics are discussed and researched (a) modularization of IFC (b) normalization of the IFC object trees and relations (c) language independency of the base data structure (d) modernization of the deployment tools and procedures for maintaining IFC. This paper reports progress on all these topics.

The normalization of the object tree is an integrated effort that involves changes in the use of objectified relations, property sets and predefined types. The modularization provides interoperability between domains and a solution to easily support incremental updates in

Berlo et al. 2021 Future of the Industry Foundation Classes: towards IFC 5

type, which can be seen as an omission. The relation between an element and its type is currently defined with the objectified relation `$(BldgDefnOnlyType)`, which is a 1:1 relation between a single element type and multiple elements.

Elements can be of a type. There can be elements without a corresponding type. As a result, in current schemas there are not only the entities `$(BldgDefnOnlyType)` and `$(BldgDefnOnlyType)`, but also an `$(BldgDefnOnlyType)` and an `$(BldgDefnOnlyType)`. The full specialization of `$(BldgDefnOnlyType)` is mirrored at the side of `$(BldgDefnOnlyType)`. These two taxonomies are not always synchronized, neither at the entity level, nor at the attribute level. For example, there is an `$(BldgDefnOnlyType)`, but no `$(BldgDefnOnlyType)`. `$(BldgDefnOnlyType)` has attributes `$(BldgDefnOnlyType)` and `$(BldgDefnOnlyType)`, but they are missing for `$(BldgDefnOnlyType)`, where they would belong instead.

Both `$(BldgDefnOnlyType)` and `$(BldgDefnOnlyType)` carry an attribute `$(BldgDefnOnlyType)`, which contains for every kind of element a value out of a domain of subtypes of that kind. The current IFC schema already contains a constraint for element is of a type to have their `$(BldgDefnOnlyType)` attribute set on the type side. For example, for an `$(BldgDefnOnlyType)` the value of `$(BldgDefnOnlyType)` can be `Window`, `ShyLight` or `Liquidator`. The value `Window` can be seen as “usual” or “normal” window. Instead of the usage of the attribute `$(BldgDefnOnlyType)` there could be the possibility to define real subtypes as `$(BldgDefnOnlyType)` and `$(BldgDefnOnlyType)`. But that will lead to an explosion of entities. And changing the domains of an enumeration is far easier than introducing new entities. We propose to make the element type mandatory, such that the specialization of `$(BldgDefnOnlyType)` is no longer necessary. A door will become an `$(BldgDefnOnlyType)` of an `$(BldgDefnOnlyType)`. As a result, more than hundred entities can be removed from the schema. But also, the attribute `$(BldgDefnOnlyType)` and `$(BldgDefnOnlyType)` can be removed from the remaining `$(BldgDefnOnlyType)`. Until now it is possible, but invalid, that a window is of a door type. This check is now made in the receiving application but is not supported by the schema. Therefore, it is not certain if this situation is recognized by the software. It is possible that one application says interprets it as a window and another as a door.

Applications which will export elements without the corresponding element type have to introduce the usage of element type, even if there is no reuse of the type. And if there are subtypes of `$(BldgDefnOnlyType)` in the schema without a corresponding type, that type must be added to the schema. Example of this is the already mentioned `$(BldgDefnOnlyType)`. When combined with moving all the subtypes out of the core, the advantage becomes more evident. An `$(BldgDefnOnlyType)` will be of the type `$(BldgDefnOnlyType)`. The `$(BldgDefnOnlyType)` will have a single attribute which has a value that defines that the element type is “door”, or one of its subtypes. Even the user-defined subtypes will move to the “late-binding”. Figure 5 shows an example with an entity instance (IT2) of class `$(BldgDefnOnlyType)` and dynamic domain class `$(BldgDefnOnlyType)`. As a result, the current “usable tree” at the “early-binding” will change into a “late-binding”. The attribute `$(BldgDefnOnlyType)` can disappear, because it will be replaced by real subtyping at the “late-binding” side.

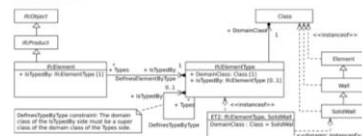


Figure 5. Provisional UML diagram showing excerpts of the core (left) and domain (right) taxonomies, classes, and associations for the occurrence object type relationship (`$(BldgDefnOnlyType)`) as well as for instance level “inheritance” (`$(BldgDefnOnlyType)`). Besides elements from the core and domain, the diagram contains a metamodel element (Class) and a population element (IT2).

Proc. of the Conference CB W09 2021, 11-19 October 2021, Luxembourg

### 3 IFC Product tree normalization

#### 3.1 New hierarchy

One of the core parts of the IFC schema is the taxonomy of “rooted” elements, a single-inheritance hierarchy of all entities outside of the resource layers, shown in Figure 4. In this taxonomy, each level follows a unique discriminator to iteratively refine the classification of entities. Level 1 contains the root of this hierarchy, conventionally named `$(Bldg)`. It has a mechanism for a stable instance identity (an attribute called `$(BldgId)`, optional at this level), a mechanism for tracking changes and status (made optional in IFC 4, named `$(BldgStatus)`) and a textual name and description. Level 2 provides the main differentiation between elements, their attributes, and relations with three subtypes of `$(Bldg)`: `$(BldgDefinition)` for things and processes, `$(BldgPropertyDefinition)` (retained) for characteristics, and `$(BldgRelationshipDefinition)` for relationships. The attribute `$(BldgId)` becomes mandatory for object definitions. These two levels are in the main unchanged from the current schema. Level 3 introduces occurrence and type for object and property definitions. Changes for this level are discussed in Section 11. Level 4 adds classification regarding shape and location.

To prevent unnecessary incompatibilities between IFC versions, the taskforce has been careful applying changes to this hierarchy. One of the main changes, that is necessitated by the shift to a partial late-bound representation of this hierarchy is a clear demarcation for the subtypes of `$(Bldg)` in Level 5 to distinguish between all things physical (`$(BldgElement)`), all elements that pertain to the spatial subdivision structure (`$(BldgSpaceElement)`) and all constructs that affect appearance of other elements (`$(BldgAppearanceElement)`). The practical advantage of this is that implementations of older versions of the early-bound schema can function well on newer versions of the late-bound schema. For example, most viewers hide `$(BldgSpaceElement)` when the model is initially loaded, and, depending on model view, need to subtract opening elements from their hosts. In this late-bound subtypes for spaces and openings can be introduced without their implied semantics from early-bound types.

The placement and specification of `$(BldgOccurrence)` and `$(BldgContent)` are still subject of discussion.



Figure 4. Overview of the new taxonomy. Note that the subtypes in this overview are not exhaustive.

#### 3.2 Occurrence and type

In IFC, all elements can be of a type. Elements can reuse information stored at the element type. Every property that is not specific for a particular instance can be stored in an element type, including the geometry. Even spatial elements can have a type, but in the current IFC version, this is only true for `$(BldgSpace)`. The `$(BldgSpace)`, `$(BldgSpace)` and `$(BldgSpace)` do not have a corresponding

Berlo et al. 2021 Future of the Industry Foundation Classes: towards IFC 5

### 3 IFC Product tree normalization

#### 3.1 New hierarchy

One of the core parts of the IFC schema is the taxonomy of “rooted” elements, a single-inheritance hierarchy of all entities outside of the resource layers, shown in Figure 4. In this taxonomy, each level follows a unique discriminator to iteratively refine the classification of entities. Level 1 contains the root of this hierarchy, conventionally named `$(Bldg)`. It has a mechanism for a stable instance identity (an attribute called `$(BldgId)`, optional at this level), a mechanism for tracking changes and status (made optional in IFC 4, named `$(BldgStatus)`) and a textual name and description. Level 2 provides the main differentiation between elements, their attributes, and relations with three subtypes of `$(Bldg)`: `$(BldgDefinition)` for things and processes, `$(BldgPropertyDefinition)` (retained) for characteristics, and `$(BldgRelationshipDefinition)` for relationships. The attribute `$(BldgId)` becomes mandatory for object definitions. These two levels are in the main unchanged from the current schema. Level 3 introduces occurrence and type for object and property definitions. Changes for this level are discussed in Section 11. Level 4 adds classification regarding shape and location.

To prevent unnecessary incompatibilities between IFC versions, the taskforce has been careful applying changes to this hierarchy. One of the main changes, that is necessitated by the shift to a partial late-bound representation of this hierarchy is a clear demarcation for the subtypes of `$(Bldg)` in Level 5 to distinguish between all things physical (`$(BldgElement)`), all elements that pertain to the spatial subdivision structure (`$(BldgSpaceElement)`) and all constructs that affect appearance of other elements (`$(BldgAppearanceElement)`). The practical advantage of this is that implementations of older versions of the early-bound schema can function well on newer versions of the late-bound schema. For example, most viewers hide `$(BldgSpaceElement)` when the model is initially loaded, and, depending on model view, need to subtract opening elements from their hosts. In this late-bound subtypes for spaces and openings can be introduced without their implied semantics from early-bound types.

The placement and specification of `$(BldgOccurrence)` and `$(BldgContent)` are still subject of discussion.

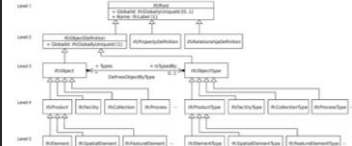


Figure 4. Overview of the new taxonomy. Note that the subtypes in this overview are not exhaustive.

#### 3.2 Occurrence and type

In IFC, all elements can be of a type. Elements can reuse information stored at the element type. Every property that is not specific for a particular instance can be stored in an element type, including the geometry. Even spatial elements can have a type, but in the current IFC version, this is only true for `$(BldgSpace)`. The `$(BldgSpace)`, `$(BldgSpace)` and `$(BldgSpace)` do not have a corresponding

Proc. of the Conference CB W09 2021, 11-19 October 2021, Luxembourg

Berlo et al. 2021 Future of the Industry Foundation Classes: towards IFC 5

### 3 IFC Product tree normalization

#### 3.1 New hierarchy

One of the core parts of the IFC schema is the taxonomy of “rooted” elements, a single-inheritance hierarchy of all entities outside of the resource layers, shown in Figure 4. In this taxonomy, each level follows a unique discriminator to iteratively refine the classification of entities. Level 1 contains the root of this hierarchy, conventionally named `$(Bldg)`. It has a mechanism for a stable instance identity (an attribute called `$(BldgId)`, optional at this level), a mechanism for tracking changes and status (made optional in IFC 4, named `$(BldgStatus)`) and a textual name and description. Level 2 provides the main differentiation between elements, their attributes, and relations with three subtypes of `$(Bldg)`: `$(BldgDefinition)` for things and processes, `$(BldgPropertyDefinition)` (retained) for characteristics, and `$(BldgRelationshipDefinition)` for relationships. The attribute `$(BldgId)` becomes mandatory for object definitions. These two levels are in the main unchanged from the current schema. Level 3 introduces occurrence and type for object and property definitions. Changes for this level are discussed in Section 11. Level 4 adds classification regarding shape and location.

To prevent unnecessary incompatibilities between IFC versions, the taskforce has been careful applying changes to this hierarchy. One of the main changes, that is necessitated by the shift to a partial late-bound representation of this hierarchy is a clear demarcation for the subtypes of `$(Bldg)` in Level 5 to distinguish between all things physical (`$(BldgElement)`), all elements that pertain to the spatial subdivision structure (`$(BldgSpaceElement)`) and all constructs that affect appearance of other elements (`$(BldgAppearanceElement)`). The practical advantage of this is that implementations of older versions of the early-bound schema can function well on newer versions of the late-bound schema. For example, most viewers hide `$(BldgSpaceElement)` when the model is initially loaded, and, depending on model view, need to subtract opening elements from their hosts. In this late-bound subtypes for spaces and openings can be introduced without their implied semantics from early-bound types.

The placement and specification of `$(BldgOccurrence)` and `$(BldgContent)` are still subject of discussion.

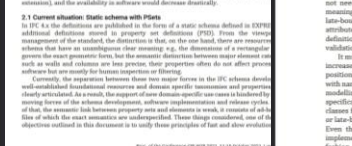


Figure 4. Overview of the new taxonomy. Note that the subtypes in this overview are not exhaustive.

#### 3.2 Occurrence and type

In IFC, all elements can be of a type. Elements can reuse information stored at the element type. Every property that is not specific for a particular instance can be stored in an element type, including the geometry. Even spatial elements can have a type, but in the current IFC version, this is only true for `$(BldgSpace)`. The `$(BldgSpace)`, `$(BldgSpace)` and `$(BldgSpace)` do not have a corresponding

Berlo et al. 2021 Future of the Industry Foundation Classes: towards IFC 5

### 3 IFC Product tree normalization

#### 3.1 New hierarchy

One of the core parts of the IFC schema is the taxonomy of “rooted” elements, a single-inheritance hierarchy of all entities outside of the resource layers, shown in Figure 4. In this taxonomy, each level follows a unique discriminator to iteratively refine the classification of entities. Level 1 contains the root of this hierarchy, conventionally named `$(Bldg)`. It has a mechanism for a stable instance identity (an attribute called `$(BldgId)`, optional at this level), a mechanism for tracking changes and status (made optional in IFC 4, named `$(BldgStatus)`) and a textual name and description. Level 2 provides the main differentiation between elements, their attributes, and relations with three subtypes of `$(Bldg)`: `$(BldgDefinition)` for things and processes, `$(BldgPropertyDefinition)` (retained) for characteristics, and `$(BldgRelationshipDefinition)` for relationships. The attribute `$(BldgId)` becomes mandatory for object definitions. These two levels are in the main unchanged from the current schema. Level 3 introduces occurrence and type for object and property definitions. Changes for this level are discussed in Section 11. Level 4 adds classification regarding shape and location.

To prevent unnecessary incompatibilities between IFC versions, the taskforce has been careful applying changes to this hierarchy. One of the main changes, that is necessitated by the shift to a partial late-bound representation of this hierarchy is a clear demarcation for the subtypes of `$(Bldg)` in Level 5 to distinguish between all things physical (`$(BldgElement)`), all elements that pertain to the spatial subdivision structure (`$(BldgSpaceElement)`) and all constructs that affect appearance of other elements (`$(BldgAppearanceElement)`). The practical advantage of this is that implementations of older versions of the early-bound schema can function well on newer versions of the late-bound schema. For example, most viewers hide `$(BldgSpaceElement)` when the model is initially loaded, and, depending on model view, need to subtract opening elements from their hosts. In this late-bound subtypes for spaces and openings can be introduced without their implied semantics from early-bound types.

The placement and specification of `$(BldgOccurrence)` and `$(BldgContent)` are still subject of discussion.

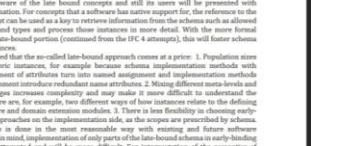


Figure 4. Overview of the new taxonomy. Note that the subtypes in this overview are not exhaustive.

#### 3.2 Occurrence and type

In IFC, all elements can be of a type. Elements can reuse information stored at the element type. Every property that is not specific for a particular instance can be stored in an element type, including the geometry. Even spatial elements can have a type, but in the current IFC version, this is only true for `$(BldgSpace)`. The `$(BldgSpace)`, `$(BldgSpace)` and `$(BldgSpace)` do not have a corresponding

Proc. of the Conference CB W09 2021, 11-19 October 2021, Luxembourg

# IDS IDS phase 1 complete

The Information Delivery Specification (IDS) had a massive commitment from industry.

Phase 1 delivered version 0.4.2. Phase 2 starts soon and is supporting vendors with consistent implementations.



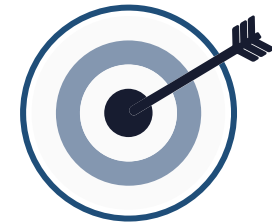
## Global participation

All major continents (and chapters) have been involved in the use-case analysis.  
~ 60 people actively involved.



## Huge market commitment

All major players committed to supporting it.  
Clear roadmap for additional features.



## Thoroughly tested

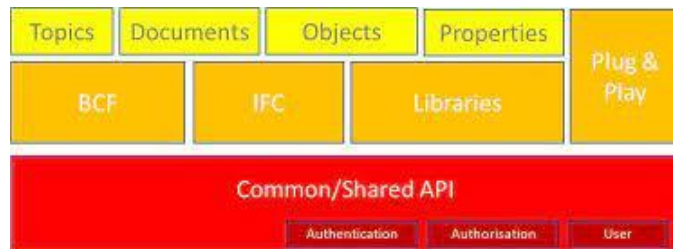
Already implemented in tools and used in practice.  
Using proven XSD restrictions & patterns.

## openCDE API portfolio

# Foundations API and BCF API update

The modularization of the APIs has continued in the openCDE program. The Foundations API agreements have been split of from BCF, creating Foundations API 1.0 and BCF (API) 3.0.

Documents API is currently in development and also based on Foundations 1.0.  
Property API is coordinating with bSDD.



### Steady progress

Strong project roadmap executed with professional recourse scheduling



### Dedicated team

Vendors building, implementing and testing APIs collaboratively.



### Use-case driven

User based testing provides input to development and roadmap

New bSDD online



The new buildingSMART Data Dictionary is up and running.

Usability driven; API first service.

Linked data functionalities (URIs, GraphQL, etc)



## Market match

Business case for end-users was the starting point.  
The new bSDD has a clear pitch of how users can save money.



## Foundation

bSDD is the foundation for other Solutions and Standards like UCMS and IDS.



## Instruct

Many hackathons, content owner workshops and test panels have been hosted to fine-tune and perfect the service.



## Trust

Next challenge is to grow tools and content. Market use is highest priority.





# bSDD Data & Information Validation



## Broad market

High prio request for IFC  
and bSDD



## Digital signature

Tag your file as 'valid'  
using a digital signature



## Insights

Data insights for future  
purposes





bSDD

# Data & Information Validation



Broad market

High prio request for IFC  
and bSDD



Digital signature

Tag your file as 'valid'  
using a digital signature



Insights

Data insights for future  
purposes

## Validation Report

### General information about the file checked

Date	2021-06-09 07:30:19
File name	IFC Schependomlaan.ifc
File size	50MB
IFC schema	IFC2X3
Authoring application	
MVD(s)	

### Overall results of the validation

Syntax
Schema
MVD
bSDD
IDS

### Metrics

Schema
MVD % compliance
bSDD % compliance

### Detailed results

- ▼ bSDD consistency
  - ▼ Domain: NL-SfB 2005
    - Classification: SPOUWWANDEN-21.12
    - Classification: FUNDATIE VOETEN-16.11
    - Classification: VLOEREN ALS GEBOUWONDERDEEL-13.22
    - Classification: FUNDATIE POEREN-16.13
    - Classification: BEKLEDINGEN-41.12
    - Classification: RECHTE STEEKTRAPPEN-24.11
    - Classification: SPOUWWANDEN-21.22
    - Classification: MASSIEVE WANDEN-22.21
    - Classification: VRIJDRAGENDE VLOEREN-23.21
    - Classification: AFVOERINSTALLATIES BUITEN HET GEBOUW-52.12
    - Classification: KOLOM-/LIGGERCONSTRUCTIES-28.11
    - Classification: AFWERKLAGE-41.11
    - Classification: ALGEMEEN-31.20
    - Classification: RAMEN DRAAIEND OP VERT. OF HOR. AS-31.24
    - Classification: CVCTEEMWANDEN-21.12

### ▼ Classification: SPOUWWANDEN-21.22

#### Requirements

IfcWall

Pset\_WallCommon - IsExternal - Boolean - TRUE

Pset\_WallCommon - LoadBearing - Boolean - TRUE

#### ▼ 1dAtUpLDz73PTe3WlpQOtG

50 % failing

Pset\_WallCommon-IsExternal-incorrect predefined value of IfcBoolean(.T.) instead of True

X

Pset\_WallCommon-LoadBearing-incorrect predefined value of IfcBoolean(.T.) instead of True

X

#### ► 1amFc\_TRT6hPyr2tODHze7

50 % failing

#### ► 3z8IK3Ks97n8dXJTJuwYh1

50 % failing

#### ► 2qrvVkep90wRmHFYUy6Epb

50 % failing

#### ► 00vrd77kP3I9I5yH8Fiv1u

50 % failing

#### ► Classification: MASSIEVE WANDEN-22.21

2<sup>nd</sup> year  
**2021**



## The year of integration and participation

Huge participation by stakeholders in  
development of standards and services.

Clear integrated development of UCMS,  
bSDD, IFC, IDS, etc.

## Preparation and Brief



## Concept Design



## Developed Design



## Technical Design



## Construction



## Handover and Close Out



**In Use**



Search B5DD

component

Search

Classifications

Properties

A 127 Classifications

81 Properties

Filter by domain

IPC 4.3

ETIM 7.0

Universal/Types 1.0

Swedish materials 1

Uniclass 2005 1.1

Plinth 1

ICC Construction 1.0

ETIM 8.0

MNVO Bridge dictionary 1.0

UCM Dams 1.0

Domain	Name	
IFC	IndrConditions	Add to IFC
IFC	QualityOfComponents	Add to IFC
IFC	FlitchAngle	Add to IFC
IFC	Platform 1	Add to IFC
IFC	SurfaceTemperature	Add to IFC
IFC	CorrosionTreatment	Add to IFC
IFC	DuctSegMethod	Add to IFC
IFC	MoistureDiffusivity	Add to IFC
IFC	LeakageClass	Add to IFC
IFC	ThermalMassHeatCapacity	Add to IFC

Exchange Information Requirements

Add custom property

Export file

Properties in EIR (4)

Name
Manufacturer
Medium
Material
pipe diameter

```

<ids:specification name="pipe fittings and pipes" necessity="required">
  <ids:applicability>
    <ids:entity>
      <ids:name>
        <xs:restriction base="<ids:entity>">
          <xs:pattern value="(IfcFlowFitting|IfcFlowSegment).*" />
        </xs:restriction>
      </ids:name>
    </ids:entity>
  </ids:applicability>
  <ids:requirements>
    <ids:property location="any" uri="http://identifier.buildingsmart.org/uri/buildingsmart/inf-4.0.0/propertyset/Pset_ManufacturerTypeInformation/<ids:simpleValue>
      <ids:name>
        <ids:simpleValue>Manufacturer</ids:simpleValue>
      </ids:name>
      <ids:value>
        <xs:restriction base="<ids:value>">
          <xs:enumeration value="Manufacturer 1" />
          <xs:enumeration value="Manufacturer 2" />
        </xs:restriction>
      </ids:value>
    </ids:property>
    <ids:property location="any" uri="http://identifier.buildingsmart.org/uri/buildingsmart/inf-4.0.0/propertyset/Pset_Pipe/<ids:simpleValue>
      <ids:name>
        <ids:simpleValue>medium</ids:simpleValue>
      </ids:name>
      <ids:value>
        <xs:restriction base="<ids:value>">
          <xs:enumeration value="cold water" />
          <xs:enumeration value="something else" />
        </xs:restriction>
      </ids:value>
    </ids:property>
  </ids:requirements>
</ids:specification>

```

[illegible]

Domain filter: NL-GB 2005

Kaliedat (PC entry): BIC044

Language filter: -

Search: Enter free search query

Name	Domain	Property
(16.2.1) kindergestruuttes leenwanden, grootwanden vanden	NL-GB 2005	Self-label
(16.2.2) kindergestruuttes leenwanden, volderwanden vanden	NL-GB 2005	Loadbearing
(16.2.3) kindergestruuttes leenwanden, gevelwanden (D02)	NL-GB 2005	
(16.2.5) kindergestruuttes leenwanden, grootwanden vanden	NL-GB 2005	
(11.2.2) buitenwanden constructief, systeemwanden	NL-GB 2005	
(11.2.3) buitenwanden constructief, spouwmuurwanden	NL-GB 2005	
(11.2.4) buitenwanden constructief, massieve wanden	NL-GB 2005	
(11.3.1) buitenwanden niet constructief, leunwanden	NL-GB 2005	
(11.3.2) buitenwanden niet constructief, systeemwanden	NL-GB 2005	
(11.3.3) buitenwanden niet constructief, spouwmuurwanden	NL-GB 2005	
(11.3.4) buitenwanden niet constructief, massieve wanden	NL-GB 2005	
(12.2.2) binnenwanden constructief, systeemwanden, vast	NL-GB 2005	
(12.2.3) binnenwanden constructief, spouwmuurwanden	NL-GB 2005	



# SCAN ME

<https://vimeo.com/615822399>

3<sup>rd</sup> year  
**2022**



## Production Stage

Still a lot to do!

Increase bSDD plugin support.

IFC 5 developments continue.

Rethinking Software Certification.

New API projects.

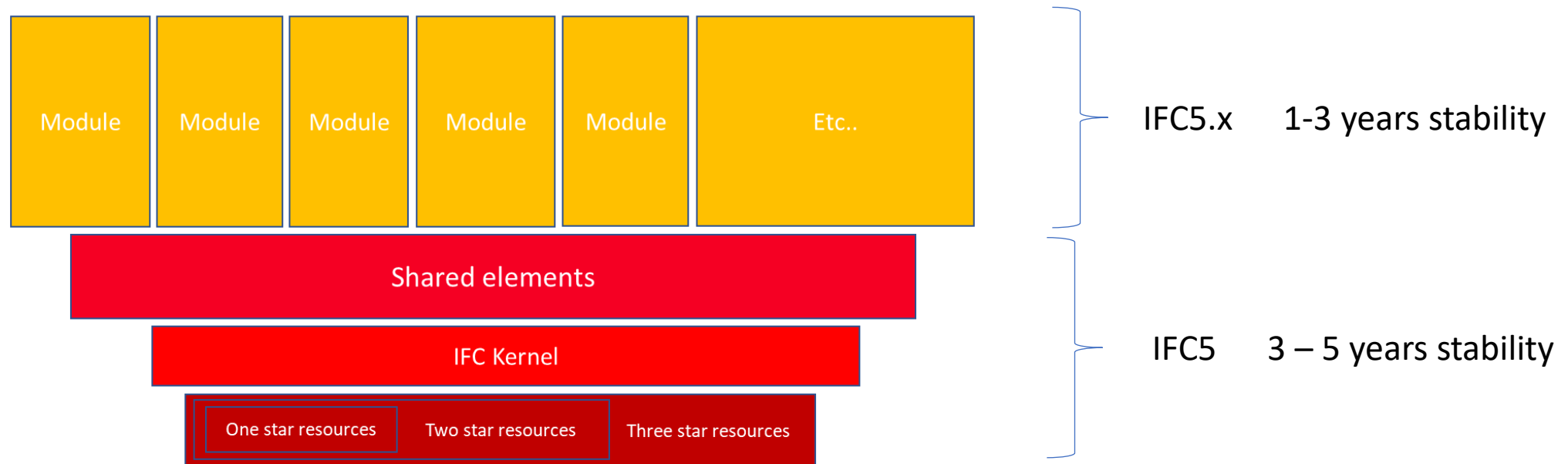
Next phase for UCMS, etc.

# IFC 5

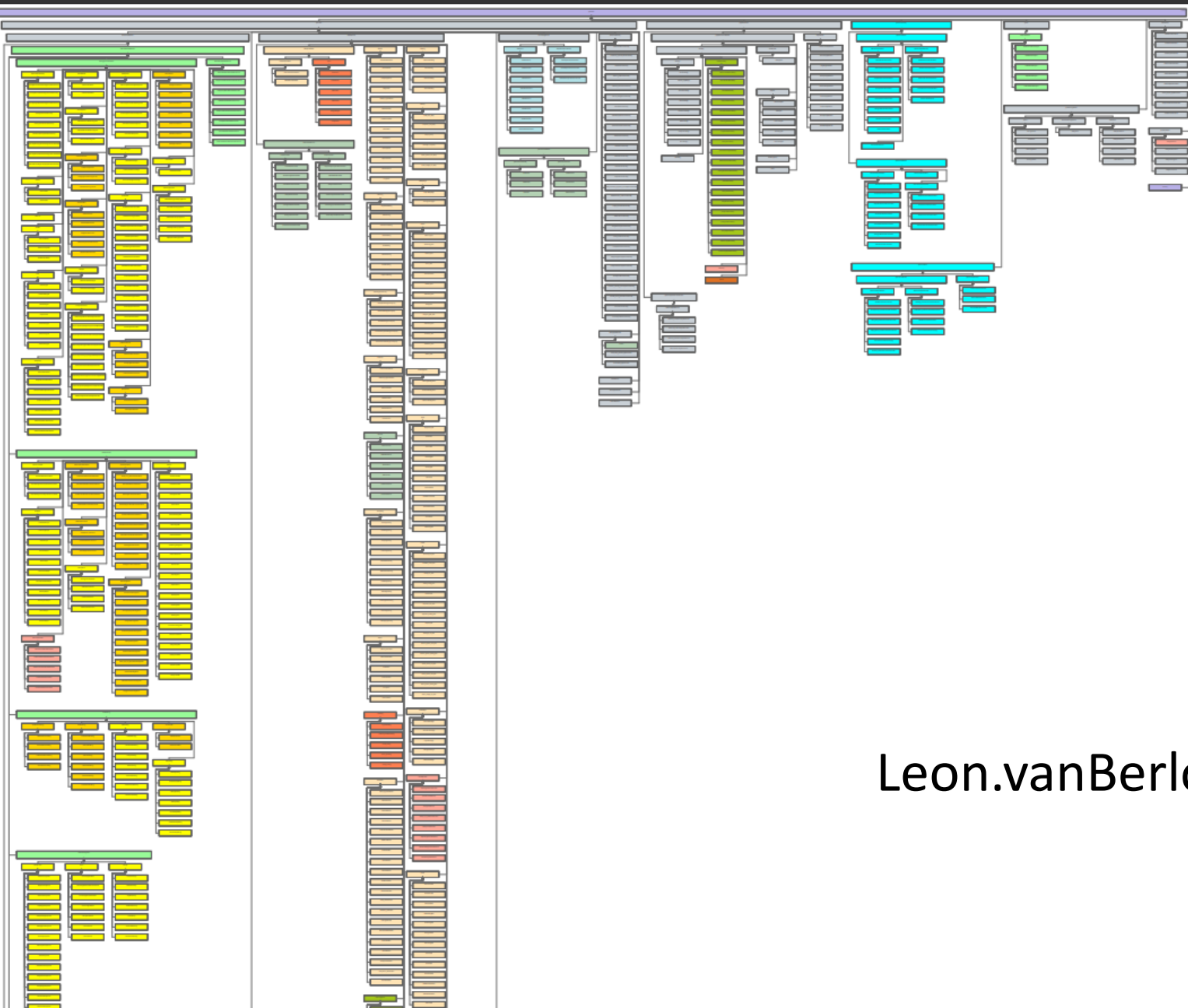
## IFC Modularization & Normalisation

Principles for IFC 5 are set out in the Technical Roadmap. First explorations can be found in the paper.

2022 will see the publication of the “IFC Precept”. A bold and drastic statement about the future of IFCs; setting direction for all future releases.







Leon.vanBerlo@buildingsmart.org

## IFC precept IFC “What if....”

2022 will see the publication of the “IFC Precept”.  
A bold and drastic statement about the future of IFCs;  
setting direction for all future releases.

IFC Precept is setting the design principles and rules  
for IFC 5 to adhere to. It is the start and the endpoint  
for IFC developments.

*“A bold and drastic statement about the future of IFCs.....”*

- Using latest STEP version (for geometry and linking files)
- Decoupling geometry and semantics
- Rethinking Software Certification
- Linking external data sources (sensors, point clouds, etc)
- New use-cases like incremental updates
- GIS integration from the core
- API and Query capabilities in mind
- Dynamically extendable
- .....

# Help!

- Write quality checks for IFC 4.3.x development (Python)
- Help script UML to XSD, ifcOWL, ifcJSON, etc..
- Make a bSDD plugin
- Help with upgrading STEP geometry core for IFC 5
- Help with upgrading STEP linked files concept
- Test/implement IDS
- Help start Property API definition / project
- Create bSDD ontology to facilitate RDF interface & SparQL endpoint
- Connect bSDD to GS1 Link resolver
- Rethink / setup Software Certification (also for BCF!)
- Develop / test digital signatures in Validation service
- IFC Precept

But also....

# We are open...

- To bcfOWL (and ifcOWL, bsddOWL, ....)
- To Parametric Geometry / IFC
- For similarity evaluation of IDSs
- To all topics on code compliance checking
- To any kind of extensions
- To changes needed for robotisation
- To blockchain & smart contract solutions
- To other kinds of geometry kernels
- To all kinds of realities (virtual, augmented, extended, etc)
- To whatever you feel a digital twin solution is
- To improving IFC for dynamic data
- To SHACL and other semantic web technologies
- To get your use-cases for incremental updates of an IFC dataset
- To adding other stuff to IFC (from inspections, pointclouds, etc)
- To automation of whatever use-case you have
- .....

# Please help!

Leon.vanBerlo@buildingsmart.org  
+31 6 423 674 65