Industry Foundation Classes - Release 2.0

Specifications Volume 2

IFC Object Model Guide

BETA - 10-January-99

International Alliance for Interoperability
Enabling Interoperability in the AEC/FM Industry

Industry Foundation Classes - Release 2.0

Specifications Volume 2

IFC Object Model Guide

Enabling Interoperability in the AEC/FM Industry

Mailing address: 2960 Chain Bridge Road - Suite 143
Oakton, Virginia 22124

Email address: IAI@Interoperability.com

Web Address: www.Interoperability.com

Document Editor

Editor	Richard See
Development committee	Specification Task Force

Document Control

Project reference	IFC Release 2.0
Document reference	IFC Object Model Guide
Document version	IFC Release 2.0 – Beta
Release date	January 10, 1998
Status	For Comments
Distribution	IAI Member Companies
Distribution format	PDF file

Revisions

Rev.	Person	Date	Description
Alpha	Richard See	10-Aug-98	Alpha release
Beta	Richard See	10-Jan-99	Beta release

Contents

1. Introduction, Scope and Assumptions

1.1 Purpose of these documents

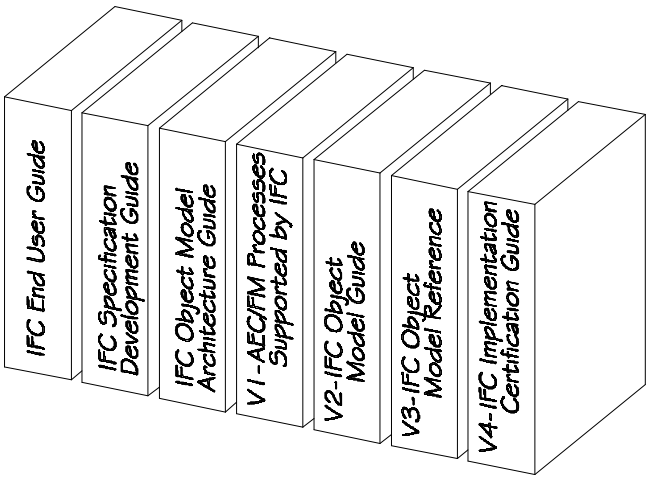
The purpose of this document suite is to provide a detailed specification of the Industry Foundation Classes (IFC) as defined by the Industry Alliance for Interoperability (IAI). The intended audience is the IAI membership, industry domain experts, and software developers interested in implementing IFC.

1.2 IFC Release Document Suite

IFC will be documented for two readers. The AEC professional and the software profession serving the AEC industry. Documents in this release include:

An Introduction to IAI and IFC

The "*An Introduction to IAI and IFC,"* as the name implies, provides AEC/FM industry professionals with an introduction to the organization, including its mission and organization. It also introduces the shared project model concept, end user benefits in using IFC compliant applications and summarizes the AEC Industry processes that are supported by this release of IFC. Finally, it provides a preview of what will be added in future releases.



IFC Specification Development Guide

The *"IFC Specification Development Guide"* defines the process used by the IAI in developing IFC. It also provides various references supporting parts of this process such as development of process diagrams, development of detailed requirement definitions and reading/creating EXPRESS (data model) definitions and EXPRESS-G diagrams.

IFC Object Model Architecture Guide

The *"IFC Object Model Architecture G*uide*"* defines the architecture used in the design of the IFC object model. This architecture is modular and layered which allows independent development and evolution of sub-schemata. This document is written for software developers who will develop applications supporting IFC.

Volume 1: AEC/FM Processes Supported by IFC

THIS DOCUMENT -- The "*AEC/FM Processes Supported by IFC"* volume documents the AEC/FM industry processes that the IFC Project Model in this release is designed to support. Therefore, this document effectively defines the scope of AEC project information included in this Release. Volumes 2 and 3 structure this information for use in applications. Note that this IFC release is limited to the information content of the foundation classes defined. Behavior for these objects, and thus the implementation of software that will support these AEC industry processes, will be defined by the implementing software vendors.

Volume 2: IFC Object Model Guide

The "*IFC Object Model Guide"* defines model design and use concepts for IFC object model. These key concepts include: an overview of model architecture, capturing design intent, sharing semantic relationships, model extension by application developers. It also describes some implementation strategies such as file based model exchange, Client-Server architectures and runtime interoperability supported through standard software interfaces of the IFC model. This includes provides an overview and example of the physical file format for file based model exchange.

Volume 3: IFC Object Model Reference

The "*IFC Object Model Reference"* provides detailed definitions for each of the classes and data types defined in the IFC object model. This includes all of the information required by the AEC processes defined in volume 1, structured in an information model detailing object class data, relationships, standard interfaces, type definitions and geometry schema use for shape representation. Additionally, it provides a data model view defined in EXPRESS and a standard interfaces view defined in IDL. Each of these code sets will be used by application developers as input into Computer Aided Software Engineering (CASE) tools to semi-automate development of applications supporting IFC. Finally, a on-line version of this information is provided using an HTML document set that is cross linked for easy access to information related to or supporting a particular class or data type.

Volume 4: IFC Software Implementation Certification Guide

The "*IFC Software implementation Certification Guide"* provides detailed information about conformance certifications issues and the methodology that will be used by the IAI to certify applications for multiple levels of IFC conformance. This includes an overview of the concepts for conformance assessment and certification, definition of various "Exchange Set" subsets of the IFC model for which certification can be assessed and an overview of the testing suites that will be used for certification testing.

1.3 Scope

1.3.1 Scope for IFC Release 2.0

Enabling interoperability between applications by different software vendors is the ultimate goal of the IAI. This is a very ambitious goal and will be achieved through a series of incremental steps.

In general, the IAI is focused on providing three things in IFC:

1. Standard definitions for the attributes associated with entities comprising an AEC/FM project model (objects)
2. Structure and relationships between these entities from the point of view of various AEC/FM professionals
3. Standard formats/protocols for two methods of sharing this information:
   - *exchange via a standard file format*
   - *exchange via standard software interfaces*

It is important to note that the software interface specifications in this release will not include any application-specific behavior. Instead, these interfaces will be limited to get and set methods for the attribute and relationship information defined in the data model.

Release 2.0 of IFC provided the infrastructure that supports this release, plus reasonable models for architecture, some HVAC, estimating, scheduling and Facilities Management. This release will build on these foundations and extend the model in several areas.

The scope for this release of the IFC Specifications is limited to:

1. Six AEC/FM domains - Architecture, HVAC engineering, codes and standards, cost estimating, facilities management and simulation
2. Only a specific subset of the processes in these domains (defined in Volume 1 of these specifications).

These domains and processes are:

Architectural Design

- *Building 'shell' design*
- *Building 'core' design*

  Stair design

- *Public toilet design*
- *Roof design*
- *Fire Compartmentation*

HVAC Engineering

- *HVAC Duct System Design*
- *HVAC Piping System Design*
- *Pathway Design and Coordination*
- *Building Heating and Cooling Load Calculation*

Codes and Standards

- *Commercial and Residential Energy Code Compliance Checking*

Cost Estimating

- *Cost Estimating*
  - *Identify Objects*
  - *Identify Tasks Needed to Install Objects*
  - *Identify Resources Needed to Perform Tasks*
  - *Quantify*
  - *Costing and Cost Summarization*

Facilities Management

- *Property Management*
  - *Enabling the use of IFC objects in property management*
  - *Grouping IFC objects*
  - *Linking the maintenance objects to the IFC objects*
- *Occupancy Planning*
- *Design of Workstations*
- *Floor Layout of Workstations for an Open Office*

Simulation

- *Photo Accurate Visualization*

All AEC domains

- *Project document management*

1.3.2 Scope of this document

This document serves as a guide to the IFC Object Model. This guide is intended to provide an understanding of the key concepts, background research, and principles used in the development of IFC. It also provides an explanation of the rationale behind the layered IFC models architecture. This layered architecture provides a framework for the evolution of the IFC model in future releases while providing stability for implementers of this release.

This information is presented in 8 sections:

1. **Introduction, Scope and Assumptions**
   Provides the reader with an introduction to the set of five volumes comprising this release of the IFC Specifications. This section outlines the information included in this document versus related documents. It will also define the scope for this release and assumptions about knowledge of the reader.

2. ***IFC Model Architecture***
   This section explains the rationale behind the layered IFC model architecture that will allow IFC to evolve in future releases.

3. ***IFC Model Overview***
   This section gives an overview of all the modules in the IFC model and can be used as a quick reference to find particular entity definitions.

4. ***Key IFC Model Concepts***
   This section presents several key concepts used in IFC which will enable much more intelligent AEC applications and which allow IFC to be extended -- in future releases, by developers and by end users. It also includes descriptions of and the rationale for using different model views - such as the EXPRESS data model view and the CORBA Interface Definition Language (IDL) view.

5. ***Guide to the Resources Layer***
   This section provides a guide to concepts in the Independent Resources Layer of the IFC Model.

6. ***Guide to the Core Layer***
   This section provides a guide to concepts in the Core Layer of the IFC Model.

7. ***Guide to the Interoperability Layer***
   This section provides a guide to concepts in the Interoperability Layer of the IFC Model.

8. ***Guide to the Domain/Application Models Layer***
   This section provides a guide to concepts in the Domain Extensions Layer of the IFC Model.

1.4 Assumptions and Abbreviations

This document assumes the reader is reasonably familiar with the following:

- AEC/FM market and project terminology
- Software industry terminology
- Concepts and terminology associated with object oriented software

The following abbreviations are used throughout the IFC Specifications:

- AEC/FM Architectural, Engineering, Construction and Facilities Management
- IAI Industry Alliance for Interoperability
- AP Application Protocol
- Arch Architecture
- CM Construction Management
- CORBA Common Object Request Broker Architecture
- COM Microsoft's Component Object Model
- DCE Distributed Computing Environment
- DCOM Microsoft's Distributed Component Object Model
- DSOM IBM's Distributed System Object Model
- FM Facilities Management
- FTP File Transfer Protocol
- GUID Globally Unique Identifier
- HVAC Heating, Ventilating and Air Conditioning
- HTTP Hypertext Transport Protocol
- IAI International Alliance for Interoperability
- IDL Interface Definition Language
- IFC Industry Foundation Classes
- ISO International Standards Organization
- FM Facilities Management
- MIDL Microsoft's Interface Definition Language
- ODL Microsoft's Object Description Language
- OMG Object Management Group
- ORB Object Request Broker
- OSF Open Software Foundation
- RPC Remote Procedure Call
- SOM IBM's System Object Model
- STEP Standard for the Exchange of Product Model Data
- TCP/IP Transmission Control Protocol/Internet Protocol
- TQM Total Quality Management
- URL Universal Resource Location

1.5 International Alliance for Interoperability (IAI)

The IAI is a 'not for profit' industry alliance of companies. Its membership is comprised of visionary companies representing all sectors of the AEC industry worldwide.

The IAI was first formed in September of 1995, by 12 industry leading companies who, during the previous year had worked together to develop proof of concept prototypes demonstrating the viability of interoperability between AEC software applications. This demonstration was shown publicly at the AEC Systems '95 conference in Atlanta, Georgia. This is the third release of IFC since that time. There are currently 50 organizations implementing software to support IFC, a number that is growing quite rapidly now.

As of this printing, the IAI includes 9 international chapters with hundreds of member companies in the following regions:

- Australasian countries
- French speaking region of Europe
- German speaking region of Europe
- Japan
- Korea
- Nordic countries of Europe
- North America
- Singapore
- United Kingdom

The IAI stated Vision, Mission and Values can be summarized as:

VISION

Enabling Interoperability in the A/E/C/FM Industry

MISSION

To define, promote and publish specifications for the Industry Foundation Classes (IFC) as a basis for information sharing through the project life cycle, globally, across disciplines and technical applications.

VALUES

- Not for profit industry organization
- Action oriented (Alliance v. Association)
- Consensus based decision making
- Incremental delivery (rather than prolonged study)
- Global solution
- Industry to define IFC
- IFC to be "open" (for implementation/use by all software vendors)
- Design for IFC to be extensible
- IFC will evolve over time
- Membership open to any company working in construction industry

2. Object Model Architecture

This subsection describes a series of concepts used in the development of the IFC Object Model. It is important to read this section before attempting to understand the model structure and content. Most elements of the model are driven from one or more of these concepts.

2.1 IFC Model Architecture Principles

The IFC Object Model Architecture has been developed using a set of principles governing it's organization and structure. These principles focus on basic requirements and can be summarized as:

- provide a modular structure to the model.
- provide a framework for sharing information between different disciplines within the AEC/FM industry.
- ease the continued maintenance and development of the model.
- enable information modelers to reuse model components
- enable software authors to reuse software components
- facilitate the provision of better upward compatibility between model releases

The IFC Object Model architecture provides a modular structure for the development of model components, the 'model schemas'. There are four conceptual layers within the architecture, which use a strict referencing hierarchy. Within each conceptual layer a set of model schemas is defined.

The first conceptual layer (shown at the bottom in **Error! Reference source not found.**) provides Resource classes used by classes in the higher levels. The second conceptual layer provides a Core project model. This Core contains the Kernel and several Core Extensions. The third conceptual layer provides a set of modules defining concepts or objects common across multiple application types or AEC industry domains. This is the Interoperability layer. Finally, the fourth and highest layer in the IFC Object Model is the Domain/Applications Layer. It provides set of modules tailored for specific AEC industry domain or application type. Additionally, this layer contains specialized model 'adapters' to non-IFC domain/application models.

The architecture operates on a 'ladder principle'. At any layer, a class may reference a class at the same or lower layer but may not reference a class from a higher layer. References within the same layer must be designed very carefully in order to maintain modularity in the model design.

Inter-domain references at the Domain Models layer must be resolved through 'common concepts' defined in the Interoperability layer. If possible, references between modules at the Resource layer should be avoided in order to support the goal that each resource module is self-contained. However, there are some low level, general purpose resources, such as measurement and identification that are referenced by many other resources.

Ladder principle expanded:

1. Resource classes may only reference or use other Resources.
2. Core classes may reference other Core classes (subject to the limitations listed in 3) and may reference classes within the Resource layer without limitations. Core classes may not reference or use classes within the Interoperability or Domain/Applications layer.
3. Within the Core layer the 'ladder principle' also applies. Therefore, Kernel classes can be referenced or used by classes in the Core Extensions but the reverse is not allowed. Kernel classes my not reference Core Extension classes.
4. Interoperability layer classes can reference classes in the Core or Resource layers, but not in the Domain/Applications layer.
5. Domain/Applications layer classes may reference any class in the Interoperability, Core and Resource layers. Additionally, classes defined within custom Interoperability Adapters (interfaces to domain or application models developed by others) may reference classes within the Interoperability layer.

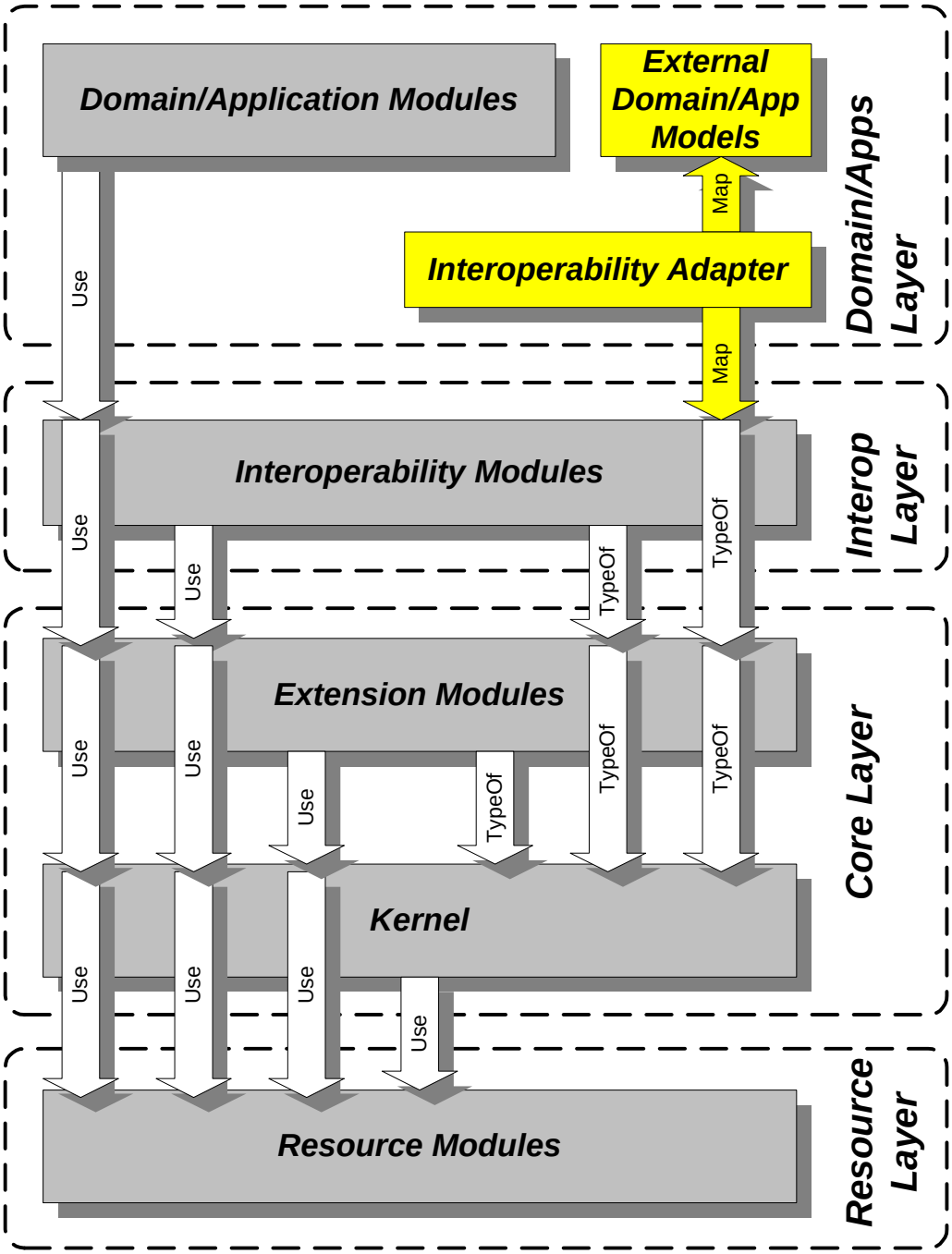


Figure 1 Layering Concept of IFC architecture

2.2 Model Modules defined in each Layer

As we saw in the last section, the IFC Model Architecture for Release 2.0 consists of the following four layers. The model modules defined in each of these layers will be introduced in this section. IFC Release 2.0 includes 24 such model modules as outlined in the diagram below.

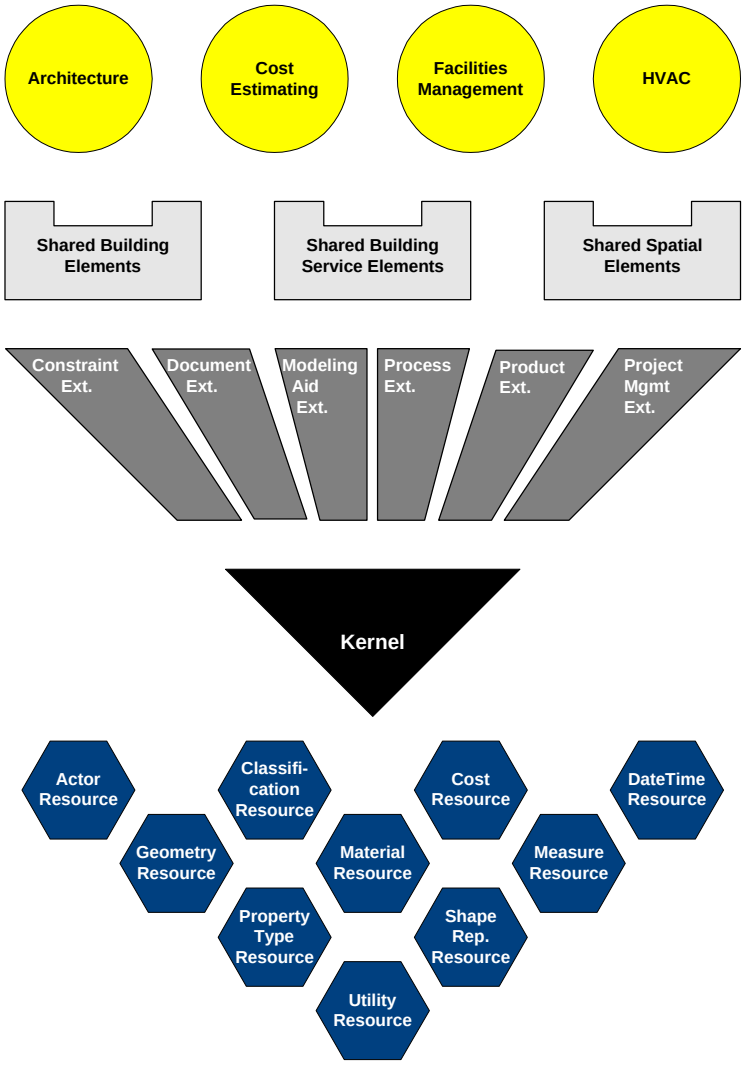


Figure 2 Model Modules defined in each layer

2.3 Resource Layer

Resources form the lowest layer in IFC Model Architecture and can be used or referenced by classes in the other layers. Resources can be characterized as general purpose or low level concepts or objects which do not rely on any other classes in the model for their existence. There are a few exceptions to this characterization. Classes from the Utility and Measure Resources are used by other, higher level resource classes.

All Resources represent individual business concepts. For instance, all information concerning the concept of cost is collected together within the cost schema, the IfcCostResource. Any classes within the Core, Interoperability or Domain/Application layers which need to use cost will reference this resource.

Similarly, all ideas concerning geometry are collected together within the IfcGeometryResource. Fundamental geometric entity definitions are defined in this resource. More specialized attribute driven geometry constructs are also defined here. Geometry will be referenced by classes defined within the Core and higher levels through the representation resource, also provided at the resource layer. However some details within the IfcGeometryResource are hidden from classes in these higher layers. There is no implication of choice for one of these representations coming from the resource layer, it simply provides the definition. A Core model object may utilize several geometry entities for representation.

2.3.1 Resource schemas for R1.5

The following resource schemas were included in IFC R1.5:

- IfcUtilityResource (object identification, object history, general purpose tables)
- IfcMeasureResource (units of measure, standard measurement types, custom measurement types)
- IfcGeometryResource (attribute driven geometric representation items, explicit geometric representation items, topological representation items, geometric models)
- IfcPropertyTypeResource (fundamental property types, property type definitions, property sets, shape representation)
- IfcPropertyResource (extended property types: material, cost, actor, classification, time)

2.3.2 Resource schemas for R2.0

In IFC Release 2.0, many of these resources were re-organized or move to separate schemas. The complete list of resources included in this release are:

- IfcActorResource (was part of IfcPropertyResource in R1.5)
- IfcClassificationResource (was part of IfcPropertyResource in R1.5)
- IfcCostResource (was part of IfcPropertyResource in R1.5)
- IfcDateAndTimeResource (was part of IfcPropertyResource in R1.5)
- IfcGeometricModelResource (was part of IfcGeometryResource in R1.5)
- IfcGeometryResource (largely the same as in R1.5)
- IfcMaterialResource (was part of IfcPropertyResource in R1.5)
- IfcMeasureResource (largely the same as in R1.5)
- IfcPropertyResource (was IFcPropertyTypeResource in R1.5
- IfcRepresentationResource (was part of IfcPropertyResource in R1.5)
- IfcTopologyResource (was part of IfcGeometryResource in R1.5)
- IfcUtilityResource (extended from R1.5)

2.4 Core Layer

The Core forms the next layer in IFC Model Architecture. Classes defined here can be referenced and specialized by all classes in the Interoperability and Domain/ Application layers. The Core layer provides the basic structure of the IFC object model and defines most abstract concepts that will be specialized by higher layers of the IFC object model.

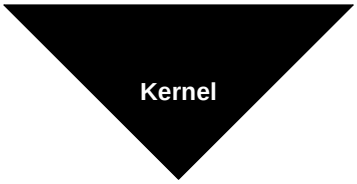
The Core includes two levels of abstraction:

1. The Kernel
2. Core Extensions

Goals for Core Model Design:

- definition of the common superset of those concepts that later can be refined and used by various interoperability and domain models
- pre-harmonization of domain models by providing this common superset
- stable definition of the object model foundation to support upgrade compatible IFC Releases

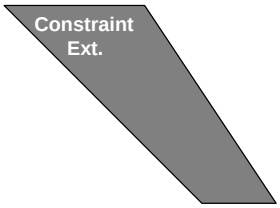
2.4.1 Kernel



The Kernel provides all the basic concepts required for IFC models within the scope of the current IFC Release. The Kernel also determines the model structure and decomposition. Concepts defined in the kernel are, necessarily, abstracted to a high level. The kernel also includes fundamental concepts concerning the provision of objects, relationships, type definitions, attributes and roles. The Kernel can be envisioned as a kind of Meta Model that provides the platform for all model extensions. The constructs that form the Kernel are very generic and are *not* AEC/FM specific, although they will only be used for AEC/FM purposes due to the specialization by Core Extensions. The Kernel constructs will be included as a mandatory part of all IFC implementations.

The Kernel is the foundation of the Core Model. Kernel classes may reference classes in the Resource layer but may not reference those in the other parts of the Core or in higher level model layers. The use of Resources will be facilitated by well defined interfaces within resource schemata. Thus, the design detail for any particular resource will be hidden from referencing classes within the Kernel.

2.4.2 Core Extensions



Core Extensions, as the name implies, provide extension or specialization of concepts defined in the Kernel. Core Extensions are therefore, the first refinement layer for abstract Kernel constructs. More specifically, they extend Kernel constructs for use within the AEC/FM industry. Each Core Extension is a specialization of classes defined in the Kernel. Figure 3 shows the further specialization of classes rooted in the IfcKernel.

Beyond this class specialization, primary relationships and roles are also defined within the Core Extensions.

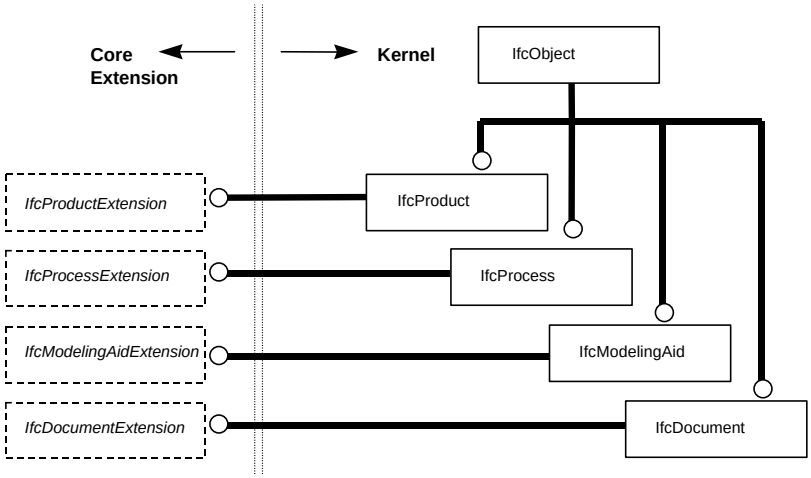


Figure 3 Core Extensions from Kernel Classes

A class defined within a Core Extension may be used or referenced by classes defined in the Inteoperability or Domain/Applications layers, but not by a class within the Kernel or in the Resource layer. References between Core Extensions have to be defined very carefully in a way that allows the selection of a singular Core Extension without destroying data integrity by invalid external references.

2.4.3 Core schemas extended from R1.5

The following schemas are included in the IFC R1.5 Core layer and extended in R2.0:

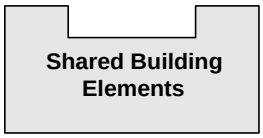
- IfcKernel
- IfcProductExtension
- IfcProcessExtension
- IfcModelingAidExtension
- IfcDocumentExtension

2.4.4 Core schemas for R2.0

Within the IFC Release 2.0 project scope the following core schemas are included.

- IfcConstraintExtension
- IfcProjectMgmtExtension

2.5 Interoperability Layer



The main goal in the design of Interoperability Layer is the provision of modules defining concepts or objects common to two or more domain/ application models. The commonly used, 'common concept' modules enable interoperability between different domain or application models. Introduction of this model layer is the best example of a general purpose model design guideline, that the model should incorporate a 'Plug-In' architecture -- allowing multiple domain or application models to be 'Plugged into' the common IFC Core. Such a 'Plug-In' architecture will also support outsourcing the development of domain/application models.

2.5.1 Interoperability schemas extended from R1.5

The following schemas were included in the IFC R1.5 Interoperability layer and extended in R2.0:

- IfcSharedBldgElements (all fundamental building elements shared between domains)
- IfcSharedBldgServiceElements (all fundamental building service elements shared between domains)

2.5.2 Interoperability schemas for R2.0

The following schemas were added to the Interoperability layer in R2.0:

- IfcSharedSpatialElements

2.5.3 Adapter Definitions

Although not yet used in the current IFC Release the concept of an 'adapter' is foreseen to access various domain models, including disperse models (i.e. those defined outside the International Alliance for Interoperability). The main requirements for Adapters are the facilitation of:

1. Direct Plug-In of IFC developed Domain Models, that is a direct reference and use of Core definitions by the appropriate Domain Models through the provision of interoperable class definitions at the Interoperability layer. This is currently the only applied technique.
2. Plug-In of externally developed, non harmonized, Domain Models via an Adapter that provides a mapping mechanism down to Core and Interoperability definitions. The definition of the Adapter Plug is in the responsibility of the Domain Model developer and is part of the Domain Model Layer.
3. Establish an inter-domain exchange mechanism above the Core to enable interoperability across domains. This includes a container mechanism to package information. Therefore an Adapter is used where the definition of the Adapter is the responsibility of all Domain Models sharing this Adapter Plug.

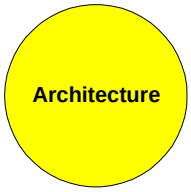
The Adapters are based on Core Extension definitions and enhance those Core Extension definitions. Those enhancements provide common concepts for all Domain Models that might further refine these concepts. As an example, the Building Element Socket provides the definition of a common wall, whereas the Architectural Domain Model will enhance this common wall with its private subtypes and type definitions within Release 3.0

time frame. An Adapter Socket that is used by several Domain Models therefore provides a medium level of interoperability through shared Adapter Socket definitions.

IFC Domain extensions that tightly couple with the Core Model such as those defined within the IFC Model (i.e., HVAC and Architecture) do not require an additional mapping of Domain Model definitions down to Core definitions, therefore they do not need specific Adapter.

Non-IFC harmonized models can be connected to the IFC Core Model through a mapping defined by a specific Adapter. This methods needs to be further elaborated within the Release 3.0 time frame. For specific high-level inter-domain exchange, that cannot be satisfied by common definitions in the Core, the Adapter may provide a specific inter-domain mapping. This Adapter type has to be developed within Release 3.0 time frame as well.

2.6 Domain/Applications Layer



Domain/Applications Models provide further model detail within the scope requirements for an AEC/FM domain process or a type of application. Each is a separate model which may use or reference any class defined in the Core and Independent Resource layers. Examples of Domain Models are Architecture, HVAC, FM, Structural Engineering etc. A main purpose of Domain Models is the provision of specialized type definitions that are tailored for the use within this domain.

Part of the Domain Model definition is the definition of the Adapter Plugs if needed. Fully harmonized IFC Domain Models will be directly plugged in the Core definitions. Domain Models which are non fully harmonized have to provide appropriate Adapter Plug definitions in order to be enabled to use the IFC model framework. The Adapter Sockets provide the guidelines to develop those Plugs. If inter-domain interoperability has to be achieved that extends the common shared Core definitions, those Domain Model developments have to be combined in order to provide an interoperable Plug.

2.6.1 Domain/Application Models extended from R1.5

The following Domain Models were included in IFC R1.5 and extended in R2.0:

- IfcArchitecture
- IfcFacilitiesMgmt

2.6.2 Domain/Application Models Added in R2.0

The following Domain Models have been added in IFC R2.0:

- IfcCostEstimatingDomain
- IfcHVACDomain

3. Object Model Overview

This section will provide a high level overview of the Object Model. It summarizes the following model modules as structured in the Model Architecture section above.

Resources layer

1. IfcActorResource
2. IfcClassificationResource
3. IfcCostResource
4. IfcDateAndTimeResource
5. IfcGeometricModelResource
6. IfcGeometryResource
7. IfcMaterialResource
8. IfcMeasureResource
9. IfcPropertyResource
10. IfcRepresentationResource
11. IfcTopologyResource
12. IfcUtilityResource

Core Layer

13. IfcKernel
14. IfcConstraintExtension
15. IfcDocumentExtension
16. IfcModelingAidExtension
17. IfcProductExtension
18. IfcProcessExtension
19. IfcProjectMgmtExtension

Interoperability Layer

20. IfcSharedBldgElements
21. IfcSharedBldgServiceElements
22. IfcSharedSpatialElements

Domain Extensions Layer

23. IfcArchitecture
24. IfcCostEstimating
25. IfcFacilitiesMgmt
26. IfcHVAC

3.1 Model Scope

Although we have focused the scope of Release 2.0 to support business processes in a selected set of AEC market domains, a large number of foundation classes are included in the Object Model. Many of these provide the underlying structure that will support an increasing scope of AEC industry processes in future releases. In this release, we have the following entity counts:

Object Model Classes			***254***
Core and Domain Extensions		**??**	
Model Structure	??		
Building Model	??		
Documents Model	??		
Process model	??		
Resource model	??		
Design aids/Design intent	??		
Objectified relationships	??		
Controls/Constraints	??		
Semantic Groupings	??		
Resources / Data Types		**??**	
Utilities resource	??		
Measure resource	??		
Geometry resource	??		
Properties resource	??		
Type definition and PropertySets	??		
Type definitions			***??***
PropertySet definitions			***??***

It is important to first understand the underlying structure of the model before looking at the individual elements.

3.1.1 IFC Object Model Hierarchy

*(** Note: this hierarchy list has not been updated in this Beta - it will be updated for the final R2 release**)*

This section provides a object class inheritence overview of the complete IFC model. It also lists the schema in which each class is defined. Detailed specifications are available for each class in the IFC Object Model Reference. These specifications include semantic definitions (for the class, attributes and relationships), software interfaces, inheritence information, type definitions, and geometry use definitions (for shape representations). Classes can be located alphabetically within the schema listed below.

	Schema	**1**	**2**	**3**	**4**	**5**	**6**	
1	IfcKernel	IfcRoot						
2	IfcKernel		IfcModelingAid					
3	IfcModelingAidExtension			IfcDesignGrid				
4	IfcModelingAidExtension			IfcGridAxis				
5	IfcModelingAidExtension			IfcGridIntersection				
	Schema	**1**	**2**	**3**	**4**	**5**	**6**	

6	IfcModelingAidExtension			IfcGridLevel				
7	IfcKernel			IfcLocalPlacement				
8	IfcModelingAidExtension				IfcConstrainedPlacement			
9	IfcModelingAidExtension			IfcPlacementConstraint				
10	IfcModelingAidExtension				IfcConstraintRelIntersection			
11	IfcModelingAidExtension			IfcReferenceGeometryAid				
12	IfcModelingAidExtension				IfcReferencePoint			
13	IfcModelingAidExtension				IfcReferenceCurve			
14	IfcModelingAidExtension				IfcReferenceSurface			
15	IfcKernel		IfcObject					
16	IfcKernel			IfcControl				
17	IfcProductExtension				IfcConnectionGeometry			
18	IfcProductExtension					IfcPointConnectionGeometry		
19	IfcDocumentExtension				IfcCostElement			
20	IfcArchitecture				IfcSpaceProgram			
21	IfcProcessExtension				IfcWorkSchedule			
22	IfcKernel			IfcDocument				
23	IfcDocumentExtension				IfcCostSchedule			
24	IfcKernel			IfcGroup				
25	IfcDocumentExtension				IfcCostElementGroup			
26	IfcArchitecture				IfcSpaceProgramGroup			
27	IfcProductExtension				IfcSystem			
28	IfcProcessExtension				IfcWorkGroup			
29	IfcProductExtension				IfcZone			
30	IfcKernel			IfcProcess				
31	IfcProcessExtension				IfcWorkTask			
32	IfcKernel			IfcProduct				
33	IfcProductExtension				IfcBuilding			
34	IfcProductExtension				IfcBuildingStorey			
35	IfcProductExtension				IfcElement			
36	IfcProductExtension					IfcOpeningElement		
37	IfcProductExtension					IfcBuildingElement		
38	IfcSharedBldgElements						IfcBeam	
39	IfcSharedBldgElements						IfcBuiltIn	
40	IfcSharedBldgElements						IfcColumn	
41	IfcSharedBldgElements						IfcCovering	
42	IfcSharedBldgServiceElements						IfcDiscreteElement	
43	IfcSharedBldgServiceElements						IfcDistributionElement	
44	IfcSharedBldgElements						IfcDoor	
45	IfcSharedBldgServiceElements						IfcElectricalAppliance	
46	IfcSharedBldgServiceElements						IfcEquipment	
47	IfcSharedBldgServiceElements						IfcFixture	
48	IfcFacilitiesMgmt						IfcFurniture	
49	IfcSharedBldgElements						IfcFloor	
50	IfcSharedBldgElements						IfcRoofSlab	
51	IfcSharedBldgElements						IfcWall	
52	IfcSharedBldgElements						IfcWindow	
53	IfcProductExtension				IfcSite			
54	IfcProductExtension				IfcSpatialElement			
55	IfcProductExtension					IfcSpace		
56	IfcProductExtension					IfcSpaceBoundary		
57	IfcKernel			IfcProject				
58	IfcKernel			IfcProxy				
59	IfcKernel			IfcResource				

	Schema	**1**	**2**	**3**	**4**	**5**	**6**	

60	IfcKernel		IfcRelationship					
61	IfcKernel			IfcRelationship1to1				
62	IfcArchitecture				IfcRelAdjacencyReq			
63	IfcProductExtension				IfcRelConnectsElements			
64	IfcProductExtension					IfcRelConnectsPathElements		
65	IfcProcessExtension				IfcRelResourceUse			
66	IfcKernel				IfcRelSequence			
67	IfcKernel			IfcRelationship1toN				
68	IfcProductExtension				IfcRelAssemblesElements			
69	IfcProductExtension				IfcRelAssemblesSpaces			
70	IfcProductExtension				IfcRelContains			
71	IfcDocumentExtension				IfcRelCostScheduleElement			
72	IfcSharedBldgElements				IfcRelCoversBldgElements			
73	IfcProductExtension				IfcRelFillsElements			
74	IfcKernel				IfcRelGroups			
75	IfcProductExtension				IfcRelSeparatesSpaces			
76	IfcProductExtension				IfcRelServicesBuildings			
77	IfcProductExtension				IfcRelVoidsElements			
78	IfcKernel				IfcRelUsesProducts			
79	IfcGeometryResource	IfcGeometricRepresentationItem						
80	IfcGeometryResource		IfcBoundingBox					
81	IfcGeometryResource		IfcCompositeCurveSegment					
82	IfcGeometryResource		IfcCurve					
83	IfcGeometryResource			IfcBoundedCurve				
84	IfcGeometryResource				IfcCompositeCurve			
85	IfcGeometryResource					Ifc2DCompositeCurve		
86	IfcGeometryResource				IfcPolyline			
87	IfcGeometryResource				IfcTrimmedCurve			
88	IfcGeometryResource			IfcConic				
89	IfcGeometryResource				IfcCircle			
90	IfcGeometryResource				IfcEllipse			
91	IfcGeometryResource			IfcLine				
92	IfcGeometryResource		IfcDirection					
93	IfcGeometryResource		IfcHalfSpaceSolid					
94	IfcGeometryResource		IfcPlacement					
95	IfcGeometryResource			IfcAxis1Placement				
96	IfcGeometryResource			IfcAxis2Placement2D				
97	IfcGeometryResource			IfcAxis2Placement3D				
98	IfcGeometryResource		IfcPoint					
99	IfcGeometryResource			IfcCartesianPoint				
100	IfcGeometryResource		IfcPolyLoop					
101	IfcGeometryResource		IfcSolidModel					
102	IfcGeometryResource			IfcAttDrivenExtrudedSolid				
103	IfcGeometryResource				IfcAttDrivenClippedExtrudedSolid			
104	IfcGeometryResource			IfcAttDrivenRevolvedSolid				
105	IfcGeometryResource				IfcAttDrivenClippedRevolvedSolid			
106	IfcGeometryResource			IfcManifoldSolidBrep				
107	IfcGeometryResource				IfcFacetedBrep			
108	IfcGeometryResource				IfcFacetedBrepWithVoids			
109	IfcGeometryResource			IfcSweptAreaSolid				
110	IfcGeometryResource				IfcExtrudedAreaSolid			
111	IfcGeometryResource					IfcAttDrivenExtrudedSegment		
112	IfcGeometryResource						IfcAttDrivenMorphedExtrudedSegment	
113	IfcGeometryResource						IfcAttDrivenTaperedExtrudedSegment	

	Schema	**1**	**2**	**3**	**4**	**5**	**6**	

114	IfcGeometryResource				IfcRevolvedAreaSolid			
115	IfcGeometryResource					IfcAttDrivenRevolvedSegment		
116	IfcGeometryResource						IfcAttDrivenMorphedRevolvedSegment	
117	IfcGeometryResource						IfcAttDrivenTaperedRevolvedSegment	
118	IfcGeometryResource		IfcSurface					
119	IfcGeometryResource			IfcCurveBoundedPlane				
120	IfcGeometryResource			IfcElementarySurface				
121	IfcGeometryResource				IfcPlane			
122	IfcGeometryResource		IfcVector					
123	IfcMeasureResource	IfcNamedUnit						
124	IfcMeasureResource		IfcContextDependentUnit					
125	IfcMeasureResource		IfcConversionBasedUnit					
126	IfcMeasureResource		IfcSiUnit					
127	IfcPropertyTypeResource	IfcProperty						
128	IfcPropertyResource		IfcCalendarDate					
129	IfcPropertyResource		IfcClassification					
130	IfcPropertyResource		IfcClassificationList					
131	IfcPropertyResource		IfcCost					
132	IfcPropertyResource		IfcDateAndTime					
133	IfcPropertyResource		IfcLocalTime					
134	IfcPropertyResource		IfcMaterial					
135	IfcPropertyResource		IfcMaterialList					
136	IfcPropertyResource		IfcMaterialLayerSet					
137	IfcPropertyTypeResource		IfcObjectReference					
138	IfcPropertyResource		IfcOrganization					
139	IfcPropertyResource		IfcPerson					
140	IfcPropertyResource		IfcPersonAndOrganization					
141	IfcPropertyTypeResource		IfcProductShape					
142	IfcPropertyTypeResource		IfcPropertySet					
143	IfcPropertyTypeResource			IfcOccurrencePropertySet				
144	IfcPropertyTypeResource			IfcSharedPropertySet				
145	IfcPropertyTypeResource		IfcSimpleProperty					
146	IfcPropertyTypeResource			IfcPropertyWithUnit				
147	IfcGeometryResource	IfcAttDrivenProfileDef						
148	IfcGeometryResource		IfcArbitraryProfileDef					
149	IfcGeometryResource		IfcCircleProfileDef					
150	IfcGeometryResource		IfcRectangleProfileDef					
151	IfcGeometryResource		IfcTrapeziumProfileDef					
152	IfcGeometryResource	IfcTopologicalRepresentationItem						
153	IfcGeometryResource		IfcConnectedFaceSet					
154	IfcGeometryResource			IfcClosedShell				
155	IfcGeometryResource		IfcFace					
156	IfcGeometryResource		IfcFaceBound					
157	IfcGeometryResource			IfcFaceOuterBound				
158	IfcPropertyResource	IfcActorRole						
159	IfcPropertyResource	IfcAddress						
160	IfcUtilityResource	IfcAuditTrail						
161	IfcPropertyResource	IfcClassificationNotation						
162	IfcPropertyResource	IfcCoordinatedUniversalTimeOffset						
163	IfcPropertyResource	IfcCostModifier						
164	IfcMeasureResource	IfcDerivedUnit						
165	IfcMeasureResource	IfcDerivedUnitElement						
166	IfcMeasureResource	IfcDimensionalExponents						
167	IfcPropertyResource	IfcMaterialLayer						

	Schema	**1**	**2**	**3**	**4**	**5**	**6**	

168	IfcPropertyResource	IfcMaterialLayerSetUsage
169	IfcMeasureResource	IfcMeasureWithUnit
170	IfcPropertyResource	IfcNotationString
171	IfcUtilityResource	IfcOwnerHistory
172	IfcUtilityResource	IfcProjectAppRegistry
173	IfcPropertyResource	IfcProjectMaterialRegistry
174	IfcUtilityResource	IfcProjectTeamRegistry
175	IfcPropertyTypeResource	IfcPropertyTypeDef
176	IfcUtilityResource	IfcRegisteredApplication
177	IfcPropertyTypeResource	IfcRepresentationContext
178	IfcPropertyTypeResource	IfcShapeBody
179	IfcPropertyTypeResource	IfcShapeResult
180	IfcPropertyTypeResource	IfcShapeRepresentation
181	IfcUtilityResource	IfcTable
182	IfcUtilityResource	IfcTableRow
183	IfcUtilityResource	IfcTransaction
184	IfcMeasureResource	IfcUnitAssignment

3.2 Resource Layer

3.2.1 IfcUtilitiesResource Schema

	1	2	3	4	5	6	
160	IfcAuditTrail						
171	IfcOwnerHistory						
172	IfcProjectAppRegistry						
174	IfcProjectTeamRegistry						
176	IfcRegisteredApplication						
181	IfcTable						
182	IfcTableRow						
183	IfcTransaction						

3.2.2 IfcMeasureResource Schema

	1	2	3	4	5	6	
123	IfcNamedUnit						
124		IfcContextDependentUnit					
125		IfcConversionBasedUnit					
126		IfcSiUnit					
164	IfcDerivedUnit						
165	IfcDerivedUnitElement						
166	IfcDimensionalExponents						
169	IfcMeasureWithUnit						
184	IfcUnitAssignment						

3.2.3 IfcGeometryResource Schema

	1	2	3	4	5	6	
147	IfcAttDrivenProfileDef						
148		IfcArbitraryProfileDef					

149		IfcCircleProfileDef					
150		IfcRectangleProfileDef					
151		IfcTrapeziumProfileDef					
79	IfcGeometricRepresentationItem						
80		IfcBoundingBox					
81		IfcCompositeCurveSegment					
82		IfcCurve					
83			IfcBoundedCurve				
84				IfcCompositeCurve			
85					Ifc2DCompositeCurve		
86				IfcPolyline			
87				IfcTrimmedCurve			
88			IfcConic				
89				IfcCircle			
90				IfcEllipse			
91			IfcLine				
92		IfcDirection					
93		IfcHalfSpaceSolid					
94		IfcPlacement					
95			IfcAxis1Placement				
96			IfcAxis2Placement2D				
97			IfcAxis2Placement3D				
98		IfcPoint					
99			IfcCartesianPoint				
100		IfcPolyLoop					
101		IfcSolidModel					
102			IfcAttDrivenExtrudedSolid				
103				IfcAttDrivenClippedExtrudedSolid			
104			IfcAttDrivenRevolvedSolid				
105				IfcAttDrivenClippedRevolvedSolid			
106			IfcManifoldSolidBrep				
107				IfcFacetedBrep			
108				IfcFacetedBrepWithVoids			
109			IfcSweptAreaSolid				
110				IfcExtrudedAreaSolid			
111					IfcAttDrivenExtrudedSegment		
112						IfcAttDrivenMorphedExtrudedSegment	
113						IfcAttDrivenTaperedExtrudedSegment	
114				IfcRevolvedAreaSolid			
115					IfcAttDrivenRevolvedSegment		
116						IfcAttDrivenMorphedRevolvedSegment	
117						IfcAttDrivenTaperedRevolvedSegment	
118		IfcSurface					
119			IfcCurveBoundedPlane				
120			IfcElementarySurface				
121				IfcPlane			
122		IfcVector					
152	IfcTopologicalRepresentationItem						
153		IfcConnectedFaceSet					
154			IfcClosedShell				
155		IfcFace					

156		IfcFaceBound				
157			IfcFaceOuterBound			

3.2.4 IfcPropertyTypeResource Schema

	1	2	3	4	5	6	
127	IfcProperty						
137		IfcObjectReference					
142		IfcPropertySet					
143			IfcOccurrencePropertySet				
144			IfcSharedPropertySet				
145		IfcSimpleProperty					
146			IfcPropertyWithUnit				
141		IfcProductShape					
175	IfcPropertyTypeDef						
177	IfcRepresentationContext						
178	IfcShapeBody						
179	IfcShapeResult						
180	IfcShapeRepresentation						

3.2.5 IfcPropertyResource Schema

	1	2	3	4	5	6	
158	IfcActorRole						
159	IfcAddress						
161	IfcClassificationNotation						
162	IfcCoordinatedUniversalTimeOffset						
163	IfcCostModifier						
167	IfcMaterialLayer						
168	IfcMaterialLayerSetUsage						
170	IfcNotationString						
173	IfcProjectMaterialRegistry						
127	IfcProperty						(IfcPropertyTypeResource)
128		IfcCalendarDate					
129		IfcClassification					
130		IfcClassificationList					
131		IfcCost					
132		IfcDateAndTime					
133		IfcLocalTime					
134		IfcMaterial					
135		IfcMaterialList					
136		IfcMaterialLayerSet					
138		IfcOrganization					
139		IfcPerson					
140		IfcPersonAndOrganization					

3.3 Core Layer

3.3.1 IfcKernel Schema

	1	2	3	4	5	6	
1	IfcRoot						
2		IfcModelingAid					
7			IfcLocalPlacement				
15		IfcObject					
16			IfcControl				
22			IfcDocument				
24			IfcGroup				
30			IfcProcess				
32			IfcProduct				
57			IfcProject				
58			IfcProxy				
59			IfcResource				
60		IfcRelationship					
61			IfcRelationship1to1				
66				IfcRelSequence			
67			IfcRelationship1toN				
74				IfcRelGroups			
78				IfcRelUsesProducts			

3.3.2 IfcDocumentsExtension Schema

	1	2	3	4	5	6	
1	IfcRoot						(defined in IfcKernel)
15		IfcObject					(defined in IfcKernel)
16			IfcControl				(defined in IfcKernel)
19				IfcCostElement			
22			IfcDocument				(defined in IfcKernel)
23				IfcCostSchedule			
24			IfcGroup				(defined in IfcKernel)
25				IfcCostElementGroup			
60		IfcRelationship					(defined in IfcKernel)
67			IfcRelationship1toN				(defined in IfcKernel)
71				IfcRelCostScheduleElement			

3.3.3 IfcModelingAidExtension Schema

	1	2	3	4	5	6	
1	IfcRoot						(defined in IfcKernel)
2		IfcModelingAid					(defined in IfcKernel)
3			IfcDesignGrid				
4			IfcGridAxis				
5			IfcGridIntersection				
6			IfcGridLevel				
7			IfcLocalPlacement				(defined in IfcKernel)

8				IfcConstrainedPlacement
9			IfcPlacementConstraint	
10				IfcConstraintRelIntersection
11			IfcReferenceGeometryAid	
12				IfcReferencePoint
13				IfcReferenceCurve
14				IfcReferenceSurface

3.3.4 IfcProcessExtension Schema

	1	2	3	4	5	6	
1	IfcRoot						(defined in IfcKernel)
15		IfcObject					(defined in IfcKernel)
16			IfcControl				(defined in IfcKernel)
21				IfcWorkSchedule			
24			IfcGroup				(defined in IfcKernel)
28				IfcWorkGroup			
30			IfcProcess				(defined in IfcKernel)
31				IfcWorkTask			
60		IfcRelationship					(defined in IfcKernel)
61			IfcRelationship1to1				(defined in IfcKernel)
65				IfcRelResourceUse			

3.3.5 IfcProductExtension Schema

	1	2	3	4	5	6	
1	IfcRoot						(defined in IfcKernel)
15		IfcObject					(defined in IfcKernel)
16			IfcControl				(defined in IfcKernel)
17				IfcConnectionGeometry			
18					IfcPointConnectionGeometry		
24			IfcGroup				(defined in IfcKernel)
27				IfcSystem			
29				IfcZone			
32			IfcProduct				(defined in IfcKernel)
33				IfcBuilding			
34				IfcBuildingStorey			
35				IfcElement			
36					IfcOpeningElement		
37					IfcBuildingElement		
53				IfcSite			
54				IfcSpatialElement			
55					IfcSpace		
56					IfcSpaceBoundary		
60		IfcRelationship					(defined in IfcKernel)
61			IfcRelationship1to1				(defined in IfcKernel)
63				IfcRelConnectsElements			
64					IfcRelConnectsPathElements		
67			IfcRelationship1toN				(defined in IfcKernel)
68				IfcRelAssemblesElements			

69				IfcRelAssemblesSpaces
70				IfcRelContains
73				IfcRelFillsElements
77				IfcRelVoidsElements
75				IfcRelSeparatesSpaces
76				IfcRelServicesBuildings

3.4 Interoperability Layer

3.4.1 IfcSharedBldgElements Schema

	1	2	3	4	5	6		
1	IfcRoot							(defined in IfcKernel)
15		IfcObject						(defined in IfcKernel)
32			IfcProduct					(defined in IfcKernel)
35				IfcElement				(defined in IfcProductExtension)
37					IfcBuildingElement			(defined in IfcProductExtension)
38						IfcBeam		
39						IfcBuiltIn		
40						IfcColumn		
41						IfcCovering		
44						IfcDoor		
49						IfcFloor		
50						IfcRoofSlab		
51						IfcWall		
52						IfcWindow		
60		IfcRelationship						(defined in IfcKernel)
67			IfcRelationship1toN					(defined in IfcKernel)
72				IfcRelCoversBldgElements				

3.4.2 IfcSharedBldgServiceElements Schema

	1	2	3	4	5	6		
1	IfcRoot							(defined in IfcKernel)
15		IfcObject						(defined in IfcKernel)
32			IfcProduct					(defined in IfcKernel)
35				IfcElement				(defined in IfcProductExtension)
37					IfcBuildingElement			(defined in IfcProductExtension)
42						IfcDiscreteElement		
43						IfcDistributionElement		
45						IfcElectricalAppliance		
46						IfcEquipment		
47						IfcFixture		

3.5 Domain/Applications Model Layer

3.5.1 IfcArchitecture Schema

	1	2	3	4	5	6	
1	IfcRoot						(defined in IfcKernel)
15		IfcObject					(defined in IfcKernel)
16			IfcControl				(defined in IfcKernel)
20				IfcSpaceProgram			
24			IfcGroup				
26				IfcSpaceProgramGroup			
60		IfcRelationship					(defined in IfcKernel)
61			IfcRelationship1to1				(defined in IfcKernel)
62				IfcRelAdjacencyReq			

3.5.2 IfcFacilitiesMgmt Schema

	1	2	3	4	5	6		
1	IfcRoot							(defined in IfcKernel)
15		IfcObject						(defined in IfcKernel)
32			IfcProduct					(defined in IfcKernel)
35				IfcElement				(defined in IfcProductExtension)
37					IfcBuildingElement			(defined in IfcProductExtension)
48						IfcFurniture		

4. Key Object Model Concepts

4.1 Specialized Views of the IFC Model

IFC can be supported through several different implementation and product alternatives. Over the next several years, we anticipate product implementations will provide the following 'categories' of functionality -- in the the order shown -- from least to most interoperable.

- Read/write of IFC model files ← Data Exchange
- Database oriented IFC model file server ← Runtime interface calls (data only)
- Runtime interoperable application objects ← Runtime interface calls (data or services)

In order to facilitate development of these different types of products and to reduce the chance for different interpretations by different vendors, we have included specialized, 'industry standard' views of the IFC model. Currently these 'standard views' include:

- Data Model for Data Exchange ← EXPRESS (ISO standard)
- Standard interface definitions ← IDL (OMG standard)

4.1.1 Data Model view in EXPRESS

EXPRESS is the ISO standard for the definition of software 'Data Models'. It is defined by ISO 10303 Part 11, "Description Methods: The EXPRESS language reference manual".

The Data Model view of the IFC object model is presented in volume 3 of these specifications - "IFC Object Model Reference".

There are serveral commercially available toolsets for compiling or interpreting EXPRESS data model definitions. Many of these implement the EXPRESS language mappings to C++, IDL, Java and the Standard Data Access Interfaces (SDAI) -- all defined in parts of ISO 10303. Others of these toolsets enable software developers to read and write ASCII files structured according to the EXPRESS schema -- using the physical file structure defined in ISO 10303 part 21.

4.1.2 Software Interfaces view in OMG IDL

The Interface Definition Language (IDL) is a standard for defining software interfaces - defined by the Object Management Group (OMG). It is most closely related to OMG's Common Object Request Broker Architecture (CORBA), and is one of the most commonly used interface definition languages in the software industry.

Within the context of IFC, we use IDL to define the standard software interfaces to be supported by IFC objects at runtime. Software vendors seeking product certification at the interface level, must successfully complete testing of these standard software interfaces.

The Software Interfaces view of the IFC object model is presented in volume 3 of these specifications - "IFC Object Model Reference".

There are several commercially available toolsets for compiling IDL interface definitions. Most of these implement the IDL language mappings to C, C++ and Smalltalk -- which helps to automate the translation from software product design (using IDL) to implementation (using one of the compiled languages listed).

4.2 Multi-Functional Elements and Systems

Many attempts to model AEC projects in the past have been significantly limited because they chose to categorize elements according to a primary functional role or as part of a system. This has not worked well for AEC projects because so many elements act in multiple roles and/or in multiple systems. In the IFC object model, we have attempted to avoid this by defining model elements, functional roles, and systems separately so that an element can assume multiple roles and/or be a member of multiple systems.

Project elements are defined as specializations of IfcElement.

As this release of IFC is limited to project information sharing only (not functional behavior), functional roles are defined as collections of attributes and relationships associated with this role that will be exposed through a software interface corresponding to that role.

Project systems are defined as specializations of IfcSystem. For this release of IFC, this specialization will be done solely through a system TypeDefinition.

See IfcElement subtypes, IfcElement.PerformedFuntions:Set [0:?] IfcElementFunctionTypeEnum and IfcSystem.

4.3 Capturing Design Intent and Design Constraints

One of the most powerful features of the IFC model design is the inclusion of entities that will allow applications to capture design intent and design constraints. The this release, we have only included a small subset of what will be possible in future releases. Nevertheless, some powerful applications functionality will be enabled, even with models defined using this release of IFC.

Some of the design intent and design constraint concepts supported in this IFC release are discussed next.

4.3.1 Specified Design Program

One of the most important information sets in any AEC/FM project is the client specified design program. Architects have developed elaborate systems for capturing this programmatic information, but to date, there are almost no applications which link these client specified design programs to design tools. We have included a small set of entities that will allow some of this design program information to be captured and related to elements in the project design. This will enable applications to aid designers in satisfying design program requirements and also in demonstrating the degree to which program criterion are satisfied.

Specifically, with this release, detailed requirements for Spaces are included as well as space adjacency requirements. This information is related directly to the spaces in the design model, thus enabling applications and/or users to verify that the client specified design program has been satisfied.

See IfcProgramGroup, IfcSpaceProgram and IfcRelSpaceAdjacency.

4.3.2 Design Modeling Aids

Another important set of design constraints which AEC professionals are currently forced to coordinate manually is design grids. Virtually all projects are designed using one or more design grids (for structure, design, planning, facilities, etc.). This release of IFC includes a set of design grid elements and alignment entities which will allow the designer to encode their intent to align building elements with design grid elements or with other building elements.

Future releases should allow much more flexible use of design Aids or constraints including more complex geometric relationships, alignment with offsets, budgetary constraints and code constraints.

See IfcDesignGrid, IfcGridLevel, IfcGridAxis, IfcGridIntersection, IfcReferencePoint, IfcReferenctCurve, IfcReferenceSurface and IfcConstrainedPlacement.

4.3.3 Connections between Model Elements

Another design intent that can be captured and communicated via the IFC model is connectivity. Current design tools do not allow such relationships and when a design change is made, the AEC user is forced to manually update the impact on elements that 'should' remain connected. With IFC models, it will be possible to capture the designer's intent to connect two or more elements. Within applications supporting this part of IFC, when a design change involves moving one of the connected elements, the application will correctly move or stretch the connected elements.

This release of IFC only supports point connections (the only subtype of IfcConnectionGeometry). However, future releases will add connections at edges and surfaces.

See IfcRelConnectsElements, IfcRelConnectsPathElements and IfcPointConnection

4.4 Relationships between Objects

4.4.1 Relationships used in this Release

This inclusion of relationships between object in an IFC model is one of the most important improvements over previous AEC software information sets. By standardizing the representation and thus the understanding of key semantic relationships between objects in IFC models, software applications will be able to deliver much more intelligent behavior in these objects.

However, the range of relationship types included in this release is limited. In general, we have included relationships that fall into five categories:

- Containment (both physical and conceptual) -- discussed below
- Grouping -- discussed below
- Connectivity -- discussed above in "Key Concepts"
- Constraint -- discussed above in "Key Concepts"
- Resource -- discussed above in "Key Concepts"
- General (where some special semantic meaning is defined) -- instance unique and not discussed

In many cases, relationships have been 'generalized' using objectified relationships -- discussed below.

4.4.2 Objectified Relationships

While more 'expensive' to implement and in terms of software performance, Objectified relationships provide several advantages over relationships declared within a specific class.

There were three driving motivations for using objectified relationsips:

1. **Generalization** (model simplification) -
2. **Many to Many relationship resolution** - In a number of cases in the model, we have situations where the relationships are many to many between two classes. Objectified relationships allow us to normalize these to a pair of many to one relationships.
3. **Relationship objects that require behavior** - In other cases, we are anticipated future requirements for the IFC project model and supporting applications. The nature of some relationships will require intelligent behavior in applications. An implementer will need to create a separate class for such a relationships in order to encapsulate this behavior. This will simplify implementation of the objects which use this relationship. As applications will be forced to objectify such relationships, we have objectified them in the object model in an effort to enable a close mapping between the shared project model and the object model used by supporting applications.

4.4.3 Containment

Throughout the model, you will see a standardized use of the relationships "HasXxx" and "PartOfXxx". These standard relationship names have been used to represent two types of Containment:

- Primary IFC model element hierarchy
- Membership in a group or system.

Primary IFC Model element hierarchy

It is important to include a primary structuring of elements in a CAD model. The IFC model structure is aligned with the most common organization of AEC project information:

Project	*>Sites*	*>Buildings*	*> Storeys*	
	>	>	>	*> Spaces*
	>	>	>	*> Elements*

That is to say; Projects contain Sites; Sites contain buildings; Buildings can contain Storeys. Additionally, Sites, Buildings and Storeys may each contain Spaces or other building elements, either directly or through another contained element.

Once again, this will enable applications to provide much more intelligent behavior. When a door or window is removed, the opening may be healed; when a room is deleted, the user may be prompted about what to do with the contained elements (assign them to another room or delete them as well).

4.4.4 Object Grouping

There are several examples of grouping elements in the model. One of the most obvious is through membership to a system object. The system object maintains a list of all the elements which are 'PartOf' that system. Possible uses for such groupings in software applications are endless.

For example, all members of a SpaceSeparation system associated with a suite of rooms could be selected for addition of sound or fire resistance attributes; all elements in an air duct distribution system could be selected for reconfiguration to rectangular versus circular shape.

This release of IFC only begins to allow such associations to be captured in the model. It does not yet include any standardized behavior that might be related to such associations.

4.5 IFC Model Extension

As the IFC Project Model must be used by a large number of applications to be successful, it is important that application developers not feel encumbered by it. In fact, it is a primary goal of the IAI that developers view the IFC Project Model as a platform which empowers them through access to a very large constituency of end users and compatible applications. Opportunities for strategic alliances, cooperative development and joint marketing with other developers should be significantly enhanced.

Therefore, we have included some concepts in the design of the IFC object Model that will enable software vendor extensions beyond the standard definitions provided by the IAI. Vendors who collaborate would be able to pass this extended information between their applications, using the standard IFC infrastructure.

Over time, such extensions should be submitted for adoption by the IAI in subsequent releases of IFC. In this way, IFC may be extended through the work of many organizations beyond the IAI.

4.5.1 Extension by Developers

In this IFC release, there are two primary extension mechanisms available to developers:

- Extension PropertySets

  → allow the vendor to define virtually any collection of data needed by their application. This data is attached on an instance by instance basis and will be preserved through round-trip data exchanges with other IFC applications.

- TypeDefinitions

  → allow the vendor to define specific object types (from the point of view of their application) which will then allow them to associate datasets which are shared by all instances of the type AND/OR vary with each occurrence of the object, but is attached to all instances as standard extension data for that type. Again, this extension data is preserved through data exchange with other certified IFC applications.

Applications seeking to use these extension mechanisms must simply implement the associated IFC model entities -- either in a data exchange or software interfaces implementation. Methods for documenting such extensions are provided in these specifications.

4.5.2 Extension by End Users

The software vendor accessible extension mechanism described above could also be exposed to the end user. The vendor would need to develop a generalized interface such that the End User can specify PropertySets that may be used in any of three cases:

- Type driven shared properties
- Type driven occurrence properties
- Extension properties

The application would also need to provide methods for the user to store and retrieve these PropertySet definitions and to associate them with individual object instances or object type definitions.

5. Guide to the Resources Layer

5.1 IfcUtilityResource

*((** this section will be completed for the R2.0 final specifications **))*

5.2 IFC Measure Resource

5.2.1 Units of Measure

5.2.1.1 Introduction

This section explains the approach to representing Units of Measure in this release of IFC.

5.2.1.2 Requirements

The following are noted requirements for Units of Measure in IFC. While not all of these will be satisfied in this release of IFC, the requirements are noted here and should be addressed over time in future IFC releases.

Quantities and Units of Measure

The schema should capture all information required to unambiguously translate dimensioned values from one units system to another.

The schema should not constrain dimensioned values to be from a single standard units system.

It is not required that the schema capture the specific scale of a unit as entered. For example, an entry of mm/s may be captured in terms of m/s. All scaling of dimensioned values for display purposes is up to the application. For example, a value of 1E3 m/s can be displayed as 1 km/s. All decisions as to how to display a value with units is made by the application and is not in the IFC scope.

The schema allows data required for an application to apply scaling to present the information with different units multipliers to be captured in the exchange data set.

The standard provides a means to capture a measurement value and its unit of measure.

Currency

Provides for capturing monetary values in terms of any defined unit of currency specified by ISO 4217.

Provides for capturing the time of effectivity of all monetary values. This may be captured for a model as whole. The time of effectivity represents the time at which all monetary values are defined. This permits conversion between currency systems based on the exchange rate in effect at the time of effectivity.

Conversions between currency units are not required to be meaningful. All interpretations of such converted values are the responsibility of the converting application.

Tolerances

The schema should provide for capture of a specified tolerance on any dimensioned value.

The schema supports expressing tolerances as a range of allowed values, a nominal value with allowed range, a nominal value with tolerance range specified in terms of a relative offset from the nominal value, and a nominal value with tolerance specified as a ratio of the nominal value.

Note: this requirement is not satisfied in the current release of IFC.

Import/Export

The schema allows translation-free exchange between applications working in the same units system.

A conforming application is required only to write an exchange file based on any of the defined IFC standard units.

5.2.1.3 Current Approaches

ISO 10303-41

Part 41 is one of the parts of the STEP Integrated Resources. Integrated resources are the fundamental semantics with which product data are exchanged. Domain-specific parts (e.g., Part 225) define domain semantics in an Application Reference Model (ARM), then define a mapping from the domain semantics to the integrated resources in an Application Interpreted Model (AIM). This must be a conformal mapping allowing unambiguous mapping from ARM to integrated resources and vice-versa.

Quantities and Units of Measure

The Part 41 approach is based on the use of ISO 1000 units as the fundamental units system, but allows great flexibility in allowing parts or applications to define additional units. These units may be:

- a composition of base SI units (e.g., meters/second),
- scaled from other units (e.g. defining inches as 25.4 mm),
- unrelated to the SI system (e.g., a unit named “parts” that would have meaning only in the ARM context) .

SI and SI-derived units are modeled with a unit vector that represents the exponent of the seven basic dimensions a unit may have (i.e., length, mass, time, current, temperature, substance, and luminous intensity). The standard allows the exponents to be real values. So, for example, a value can be expressed in units of $m^{1.5}/s^{3.777}$. Real values may been chosen as the most general, but ARMs can restrict values to integer as necessary.

The schema allows arbitrarily complex derivations. For example, an application can define a unit called “foo” as m/s^2, a unit called “bar” as foo^2/s, a unit called “baz” as foo^3/bar^2, ad infinitum. This allows any arbitrary units to be so defined. Note that these derived units are not directly referenced back to the fundamental dimension vector. This permits the recovery of the name of the input unit, perhaps capturing some essence of intent.

Conversion units allow scaling only. This may not support the mapping of one unit to another where a constant offset is required (e.g., degrees Fahrenheit mapped to degrees Celsius).

The entity global_unit_assigned_context establishes a units system (a set of units) that is then used within a specific context. An ARM would define this context at appropriate points. An ARM can decide to have a single context, or can arrange nested contexts within some scope hierarchy. The context is a set of units and can include any mix of SI, SI-derived, converted, and context-dependent units. A receiving application is be required to be capable of recognizing all possible units and performing conversions as required between them. Alternatively, an ARM can define rules on the context so that a constrained set of predefined units is all that can be exchanged.

Currency

This standard does not define any standard currency units.

Tolerances

This standard does not address tolerances on values.

Import/Export

This standard does not apply any a-priori constraints on the units that an exchange data set may be conveyed in. ARMs are presumably responsible for defining these constraints.

5.2.1.4 Units in IFC

Quantities and Units of Measure

The IFC measure resource contains the same (or similar) semantics as Part 41.

Domain model developers may define specialized subtypes of IfcMeasureWithUnit for each standard quantity in the IFC domain (i.e., enumerate the quantities of the IFC domain). This can be done by defining constraints on the dimension vector to ensure consistency. For example:

```
ENTITY MassFlowrate
   SUBTYPE OF (IfcMeasureWithUnit);
WHERE
   wh1: derive_IfcDimensionalExponents
   (SELF\IfcMeasureWithUnit.UnitComponent)
   = IfcDimensionalExponents(0,1,-1,0,0,0,0);
END_ENTITY;
```

Attribute types in IFC domain schemata are defined using these types. For example:

```
ENTITY PumpSpecification;
   MaximumFlowrate : IfcMassFlowrate;
END_ENTITY;
```

Within an application that defines such subtypes, any defined unit can be assigned to the value instance. The “where” rule ensures that any unit assigned to the value is a mass flow rate (mass/time) and allows the exchange data set to be checked for consistency. This release of IFC does not include definition of any such specialized units of measure. However, this will be considered in future releases.

Currency

```
TYPE IfcCurrencyTypeEnum = ENUMERATION OF (
      AED,  AES,  ATS,  AUD,  BBD,  BEG,  BGL,  BHD,  BMD,
      BND,  BRL,  BSD,  BWP,  BZD,  CAD,  CBD,  CHF,  CLP,
      CNY,  CYS,  CZK,  DDP,  DEM,  DKK,  EGL,  EST,  FAK,
      FIM,  FJD,  FKP,  FRF,  GBP,  GIP,  GMD,  GRX,  HKD,
      HUF,  ICK,  IDR,  ILS,  INR,  IRP,  ITL,  JMD,  JOD,
      JPY,  KES,  KRW,  KWD,  KYD,  LKR,  LUF,  MTL,  MUR,
      MXN,  MYR,  NLG,  NZD,  OMR,  PGK,  PHP,  PKR,  PLN,
      PTN,  QAR,  RUR,  SAR,  SCR,  SEK,  SGD,  SKP,  THB,
      TRL,  TTD,  TWD,  USD,  VEB,  VND,  XEU,  ZAR,  ZWD);
END_TYPE;
```

The Part 41 unit selection type is extended to include currency_unit.

Note that the above does not allow currency units to be used in derivations. It also does not allow currency units to be composed with other units.

Tolerances

Tolerances on values are not supported in this release of IFC.

Import/Export

Applications can exchange data in any defined set of units defined in accordance with the above units schema.

Applications are required to include the definition of all units used in the exchange data in the exchanged data set using the UnitsInContext attribute on the IfcProject object (of type IfcUnitAssignment).

Writing applications may write data sets using the same units used in the application.

Reading applications can examine the defined units set and determine translation requirements. The unit definitions allow unambiguous translation. If the reading application uses the same units as the writer no translation will be required, even if the units are not SI units.

5.3 IfcGeometryResource

5.3.1 Geometry

5.3.1.1 Introduction

The IFC Object Model includes geometry definitions for multiple purposes. In general, we have chosen to use an implicit geometry definition of the physical shape of an object. In addition to this, applications may optionally associate explicit geometry representations using an adapted subset of the entities defined in STEP Integrated Resource part 42 for Geometric and Topological Representation (see references).

In order to allow for the coordination of multiple geometry representations, we have included the concept of a Reference Geometry. In this release, we have chosen to limit the use of reference geometry to a single placement entity (location and orientation), adapted from STEP part 42. There is a single placement defined for any object. It is 'used by' the Bounding Box, Implicit and optional Explicit geometry representations.

Finally, in cases where specific geometry definition are not provided, a general purpose BoundingBox representation is available for use with any physical object. This BoundingBox can thus be used by any application as the minimal geometry representation for any object, even if a more specific representation is available. The BoundingBox representation has been made mandatory for any element which has geometry so that all IFC applications can rely on it to provide location, orientation and extent.

5.3.1.2 Scope

This section defines placement, minimal BoundingBox geometry, Implicit geometry (called Attribute Driven or AttDriven geometry in this release) and Explicit geometry resources used to define the shape and spatial arrangement of IFC project elements.

Such information can be used at all stages in the life-cycle of a building including: design process, construction, facilities management and operations. The purpose of this section is to enable software applications in all building and construction industry sectors to exchange building element shape and spatial arrangement information.

The following are within the scope of this part of the specifications:

- Implicit (Attribute Driven) representation of the 3D shape of building elements
- The spatial arrangement of building elements that comprise the assembled building

The following are outside of the scope of this part of the specifications:

- Symbolic representations
- The contents of building standards
- Specifications of properties of building elements, including material composition
- Association of properties and classification information to building elements
- The assembly process, joining methods, and detailed connectivity of building elements
- Approval, revision, versioning and design change histories

5.3.1.3 Definitions and Abbreviations

The following definitions apply to this section:

- **explicit geometry**: A geometry representation void of semantic meaning for its parts. Pure geometry definition in terms of points, curves, surfaces and solid primitives. Examples: a cube could be defined in terms of eight points, 12 edge curves, 6 bounded surfaces or some combination.
- **implicit geometry** or **Attribute Driven**: A geometric representation driven by attributes. Such a representation will have few (if any) constraints. For example, a cube can be defined using a placement entity (see placement entity definition) and a length attribute (aligned with the X-Axis of the local coordinate system), a width attribute (aligned with the Y-Axis of the local coordinate system) and a height attribute (aligned with the Z-Axis of the local coordinate system) -- length, width and height being the "driving" attributes.
- **parametric geometry** or **constrained geometry**: A geometry representation driven by functions; complex geometry reduced to simple parameters which may include arbitrarily complex (external) constraints between objects. Parametric geometry can be defined using either implicit or explicit geometry methods. For example, a cube defined using implicit geometry would have constraints applied to the length, width, or height attributes based upon adjacent objects or design criteria.
- **reference geometry** and **placement entities**: Defines the most fundamental elements of the geometry for an object - which allow coordination of multiple geometry representations (e.g. plan view, section view and 3d shape represenations). An example of a reference geometry is an oriented vertex, which consists of a 3D Cartesian point placement entity and a direction placement entity, which specify a local coordinate system fixed at a particular location in Cartesian space. Refer to the sections titled *Reference Geometry and the Bounding Box* and *Geometric Primitives for IFC Geometry* for more information on reference geometry and placement entities.
- **bounding box**: Defines the extents of the shape geometry for an object. A bounding box is an octahedral boundary element defined by its length attribute (aligned with the X-Axis of the local coordinate system), width attribute (aligned with the Y-Axis of the local coordinate system) and a height attribute (aligned with the Z-Axis of the local coordinate system).

The following abbreviations may be used in this section:

- B-rep Boundary representation
- CSG Constructive Solid Geometry
- LCS Local Coordinate System

5.3.1.4 Reference Geometry and the Bounding Box

Reference geometry is the mechanism used to coordinate multiple geometric representations. For this release of IFC, reference geometry is limited to placement of an element. This placement is always relative to a reference element, which enables relative placement.

Over time, IFC objects that have geometry must be able to accommodate multiple geometric representations or views. For example, an object may have a different representation depending upon the phase of the project. Similarly, the architect may choose to view an object differently from an HVAC engineer. These multiple representations of objects will all utilize the same reference geometry. If an application changes the reference geometry, then other applications are responsible for updating their views to reflect these changes.

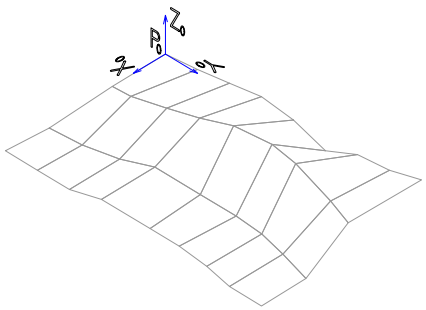


Figure 5-1: IfcSite object

One of the consequences of a single reference geometry with multiple shape representations that reference it is that the local coordinate systems of the different shape representations are consistent. Each physical IFC object with different shape representations has one positioning entity which is valid for all views of this IFC

object. From an implementers point of view, this means that the same transformation matrix is applied to all shape representations related to a given object. This mechanism is applied whether a shape representation is defined implicitly, explicitly or parametrically.

Each object that has a geometric representation will carry both a reference geometry (in this Release - limited to placement) and a bounding box.

There are three general types of reference geometry: placement, open path, and face. In IFC release 1, we have only used placement. Future releases of IFC will make use of other forms of reference geometry.

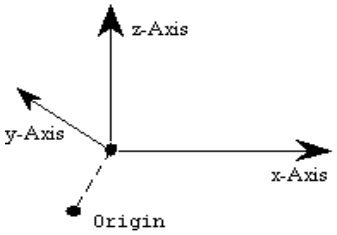
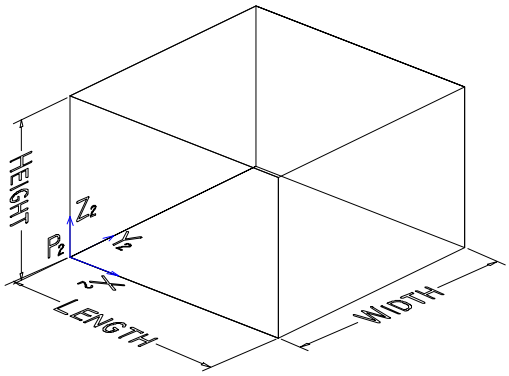


Figure 5-2: Reference Geometry entity

The IfcPlacement entity is defined by a 3D cartesian_point entity which fixes the location of the object's geometry representation in 3D space. This point is defined relative to the placement of a reference entity in all cases except IfcSite, which is defined relative to a reference global position defined by longitude, latitude and elevation. The reference entity can be any other project object. This allows users and applications to arrange relative placements that will simply an modifications to a related group. For example, if the contents of a Space are placed relative to that Space, moving the Space will automatically result in a like movement of the contents.

IfcPlacement also includes one or more direction entities which define an orientation about it's location. The combination of this location and orientation defines a local coordinate system for the object. This local coordinate system is used to define the shape representations of the object, and all geometric primitive references will be relative to this local coordinate system.

Every object with geometry are required to have a minimum default representation of a bounding box. The bounding box is the one representation that will always exist and be available. Even if more specific representations are associated with an object, the BoundingBox should be updated and made consistent so that applications which may only want this minimal representation will have a valid view of the object geometry.



The bounding box describes the object's extents with a length attribute (associated with the X-Axis of the reference geometry's LCS), width attribute (associated with the Y-Axis of the reference geometry's LCS) and a height attribute (associated with the Z-Axis of the reference geometry's LCS).

Please see the Implicit geometry example for the Light Post below for an example definition of a bounding box.

5.3.1.5 Implicit Geometry Representation

Introduction

As described in the introduction to this section, the preferred definition of geometry used to represent the shape of IFC objects will use implicit (or Attribute Driven) geometry. This can be thought of a "simple parametric" geometry.

Few projects to date have attempted to exchange geometry information using this approach. A notable exception was the NICC project in Sweden. In studying the projects which have attempted to use implicit geometry and in analyzing the way geometry can be created by most CAD systems, we have observed two consistent themes:

1. Use of a set of predefined geometry primitives
2. Use of three geometry creation methods for defining geometry implicitly:
   - extrusion: surfaces created through extrusion of a profile along a path

- revolution: surfaces created through rotating a profile about an axis
- composition: solids or surfaces created through the composition of multiple sub-parts

Each of these is an optimal approach for describing (and creating) certain types of geometry representations. Together, these primitives and methods for generating more complex geometry provide an adequate toolset for describing the geometry of any IFC object type in AEC.

In order to define the geometry of IFC objects parametrically, we will:

1. parameterize a set of geometry primitives widely supported in the industry
2. propose a notation system which supports use of these primitives in extrusion, revolution, and composition.

In the next section we will introduce use of this notation system. The sample definition includes three parts:

1. Implicit Geometry placement - *as described in the section above*
2. BoundingBox geometry - *as described in the section above*
3. Implicit geometry definition - *using primitives, extrusion, revolution and composition*

It is very important to note that the “simple parametric” approach that we are using to define implicit geometry means that the information to be stored in the IFC Project Model is the parameters for the construction of the geometry, NOT THE RESULTING GEOMETRY ITSELF. This means that an application which supports IFC must construct the geometry representation that is appropriate (for that application) using these parameters and the definitions in the IFC Object Model specification.

The approach used here makes use of concepts first introduced in the NICC project in Sweden and the High Level Interface (HLI) defined by IEZ.

Implicit geometry representation classes

We have included 3 general groups of explicit geometry representation classes. These are used in the definition of a product shape using implicit geometry.

- *Attribute driven profiles*
- *Attribute driven extruded solids*
- *Attribute driven revolved solids*

Sample implicit geometry representation

*((** this section will be completed for the R2.0 final specifications **))*

5.3.1.6 Explicit Geometry Representation

Introduction

In the event that the Implicit geometry shape, as defined in the previous section, is not adequate for some applications' needs, an optional explicit shape definition may also be attached to an IFC object.

We have adapted parts of STEP part 42 to define the Explicit Geometry sections of the geometry resource.

Explicit geometry representation classes

We have included 4 general groups of explicit geometry representation classes. These are used in the definition of a product shape using explicit geometry

- *Component Shape Representation*
- *Site Shape Representation*
- *Space Shape Representation*
- *Bounding Box*

Sample explicit geometry representation

*((** this section will be completed for the R2.0 final specifications **))*

5.3.1.7 References

ISO 10303-42:1995, Industrial automation systems and integration - Product data representation and exchange - Part 42: Integrated Generic Resources: Geometric and Topological Representation.

Speedikon High Level Interface Version 3.0: The Interface of IEZ AG Bensheim (1995), IEZ Bensheim.

Tarandi, V (1993), Object oriented communication with NICC (Neutral Intelligent CAD Communication), in Management of Information Technology for Construction, World Scientific & Global Publication Services, 1993, Singapore, pp. 517-527.

5.4 IfcPropertyResource

5.4.1 Classification

5.4.1.1 Introduction

The IfcClassificationFunction model has been developed from that contained within the proposed ISO 10303 part 106 (Building Construction Core Model) which was built in conjunction with ISO Technical Committee 59. It represents an agreed data model for the classification of objects. In developing the model, it was recognized that there are many different classification systems in use throughout the industry and that their use differs according to geographical location, industry discipline and other factors. For a generic model such as the IFC Integrated Model, it is necessary to allow for the adoption of any rational classification system whether it be based on elements, work sections or any other classifiable division.

5.4.1.2 Scope

This part of the Industry Foundation Class Specifications 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 this part of the specifications:

- The provision of one or more classifications to an object.
- The designation of a classification in terms of its author, table and notation.

- The provision of a means of semantically identifying the meaning of a classification notation.
- The identification of the classification which is the most relevant at a particular time.

The following are outside of the scope of this part of the specifications:

- The ability to translate from one classification notation to another.

5.4.1.3 Background

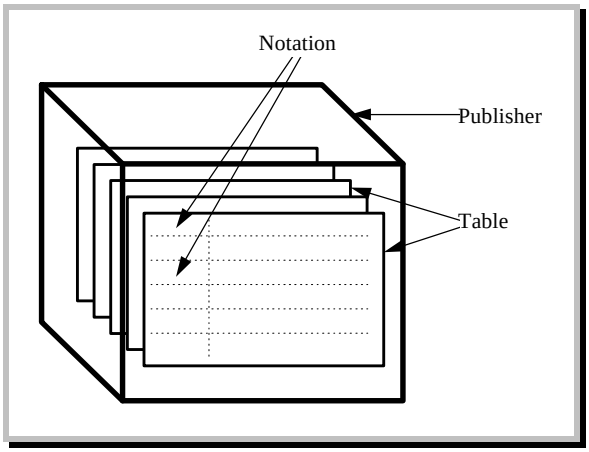
The principal applied is that any type of object can be classified. For the purposes of classification, no distinction is made between a product, a process, a control or a resource. Any of these types may be dealt with via classification tables. To satisfy the classification requirements, one or more IfcClassificationFunctions are used. It should be noted that the IFC Integrated Model specifically allows for more than one classification function to be applied to an object. The model also allows for the classification requirement to be satisfied by zero classification functions and this allows for situations where classification is not required.

Equally, and as would be expected, a classification function can be used to satisfy the classification requirements of many objects since there are likely to be multiple instances of the same class of object.

An IfcClassificationFunction is derived from a table of CharacteristicFunctions. This is the classic table to be found in all classification systems. The model allows for the IfcClassificationFunction to comprise a list of CharacteristicFunctions. The use of the ‘list’ aggregation in the relationship rather than ‘set’ implies that there is order in the CharacteristicFunctions. This order is used by allowing the IfcClassificationFunction to have a priority value. This is an integer which points to the index of the CharacteristicFunction in the list which has the highest priority. The term 'priority' is used rather than any other, such as index, to capture the idea that at any one time a particular characteristic function has more importance than other characteristic functions which may be available. Index as a term would not capture this concept of importance. By changing the priority value within an application, the classified view of an object is allowed to change to suit prevailing circumstances.

To allow the classification system used to be recognized, each CharacteristicFunction has attributes which define the publisher, the element table and the notation. The publisher identifies what would usually be considered to be the name of the classification system such as CI/SfB, BSAB, CAWS, Masterformat, Uniformat etc. whilst the element table determines which of the various forms or tables within the system is used. Notation identifies the classification reference normally used. For instance, within the CI/SfB system piped engineering services are contained under the 500 notation whilst in the CAWS system they are within the S-- notation. A further attribute available is a textual description of the classification notation so that, as well as the actual notation, a semantic idea of the notation meaning can be shared.

These attributes of the CharacteristicFunction can be visualized from the card index box shown here. The publisher is the box, which contains various cards which are the tables, each table having a set of rows with each row being a notation.



It is important to note that, whilst several different classifications may be applied to an object via the classification function, the model does not imply that there is any equivalence between such classification notations. This precludes the use of the model as a means for the translation of 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.

5.4.1.4 References

ISO 10303-WD106 (version S511);, Industrial automation systems and integration - Product data representation and exchange - Part 106: Building Construction Core Model

5.4.2 Cost

5.4.2.1 Introduction

The IfcCost model provides a facility which enables the exchange or sharing of basic information about the cost of objects. It does not attempt to develop ideas of cost beyond those of simple description, quantity and monetary amount, leaving these to be developed in more detail elsewhere.

5.4.2.2 Scope

This part of the Industry Foundation Class Specifications specifies the use of the independent resources necessary for the scope and information requirements for the exchange of building cost information between application systems. Such information may be used at all stages of the life-cycle of a building, but is intended primarily to support the construction stage and its need to develop costs for budget, tender variation and interim payment application purposes . The purpose is to enable software applications in all building and construction industry sectors to exchange building element shape and spatial arrangement information.

The following are within the scope of this part of the specifications:

- The provision of a cost as a monetary amount to an object together with the currency in which that amount is designated
- The provision of a list of cost factors which may be applied to a cost to vary its amount together with the purpose of such cost factors.
- The designation of a cost in unitary terms in cases where there may be many objects with the same cost, or itemized terms in cases where cost is applied to singular objects.

The following are outside of the scope of this part of the specifications:

- The detailed development of cost schedules.
- Assignment of cost to transactions recorded on documents such as interim applications for payment, variation orders etc.
- Costs recorded in multiple currencies.
- The calculation of cost through the application of cost factors to a cost amount.
- The distinction between cost additions and cost deductions other than through allowing cost factors to have both positive and negative values.

5.4.2.3 Background

Every cost is considered to be delivered in a monetary amount which is subject to a denomination of currency applicable to the project. A series of cost factors may be applied to a cost such that, from a base cost (such as that which may be quoted in a manufacturers or suppliers price list), an actual cost may be calculated Note that cost factors may be applied either as a positive or negative quantities. Note that the cost factor relation is a list so that, if more than one cost factor is applied, an order of applying the cost factor is implied. Thus, if the base price is initially stored as cost from a price list whereas in practice, that price is subject to a 15% uplift for one price list amendment, a 10% uplift for a subsequent price list amendment, a 25% trade discount to the user and a 5% rebate on net cost for bulk purchases over a period of time by the user the order of application would be +15%, +10%, -25%, -5%. Clearly, this order is important in deriving an actual cost from a base cost. Application of the cost factors in a different order would give a different, and incorrect, figure. In addition to the cost factor quantity, provision is made for the attachment of a cost factor purpose so that the rationale of each cost factor is identifiable.

Costs may be identified either as UnitCost or ItemCost or as both (no exclusion constraint is applied to these subtypes of cost since it may be appropriate to identify costs in both unit and item quantities). These reflect the way in which they would be applied normally. A UnitCost would be expected to be multiplied by the quantity of an item to which it is applied whereas an ItemCost would be the cost for a whole item (usually subject to a quotation).

5.4.2.4 References

ISO 10303-WD106 (version S511);, Industrial automation systems and integration - Product data representation and exchange - Part 106: Building Construction Core Model

5.4.3 Identification

5.4.3.1 Introduction

The IFC Integrated Model provides a rich range of identification possibilities for varying purposes. Identification is required to manage the process of what and where an object is and to provide some terminal information concerning status so that users and software applications know with which object they are dealing.

5.4.3.2 Scope

This part of the Industry Foundation Class Specifications specifies the use of the independent resources necessary for the scope and information requirements for the exchange and sharing of object identification information between application systems. Such information will be used at all stages of the life-cycle of a building

The following are within the scope of this part of the specifications:

- The provision of an identification to an object which allows it to be consistently understood as the same object irrespective of the systems which may share in its use.
- The provision of an identification to an object which allows it to be identified in terms of its physical existence in reality (that is, an identification which remains with the physical object wherever it moves).
- The provision of an identification to an object which allows it to be identified in terms of its logical existence at a location in space (that is, an identification which remains with the place at which a physical object is located irrespective of the physical object located at that place).
- The provision of an identification to an object which allows the identification of the software application which created it.
- The provision of date identification which allows both the creation date and the deletion date of the object to be identified.

The following are outside of the scope of this part of the specifications:

- The provision of any specific identification which may be applied to many instances of the same type of object other than as may be provided for by the logical identification facility.
- Status identification other than terminal status designated by creation and deletion.

5.4.3.3 Background

Every object must have an identification. This is an inviolable rule of the IFC Integrated Model. Identification allows the progress of an object to be traced in various ways. Tracing may be for many purposes. Depending on the purpose, a particular form of identification may be mandatory or optional.

Each object must have a unique identifier. This remains with it as an invariant property and allows it to be recognized across different systems which may impose their own internal identifications as well. The unique identifier is absolutely necessary for shared data use and also for the possible development of incremental data exchange using exchange files. Without such an identifier, it would not be possible to recognize individual objects across exchanges.

Each object may potentially have a physical identifier. This is data which is normally associated with a physical product and which again is expected to remain with that physical item throughout its usage. It is the equivalent to a serial number on the nameplate of a mechanical or electrical device.

Note that the physical identifier is associated with the physical item which is not the same thing as an instance of an entity. It is possible for a physical item to be replaced by a different physical item but for it to remain the same instance. The proposed ISO 10303-221 demonstrates this principle in its discussion of pumps.

The provision is also made for objects to have logical identifiers, a relationship of zero, one or many being allowed. A logical identifier is required to have both a logical_id_value and may optionally be given a logical_id_purpose which determines its purpose.

A logical identifier is not required to be unique whereas both the unique identifier and the physical identifier are. By not requiring that the logical identifier is uniquely associated with an instance of an object, it is possible to use it in a flexible way. For instance, it might be used for asset identification where a number of instances of a physical product form a single asset. At the same time, a different logical identifier might be used for scheduling purposes, for example, identifying that all light fittings with a given identification are of the same type.

Each object is required to have an application identifier which specifically identifies the software application which created it. This is accompanied by an object creation date which indicates the date on which the object was first instantiated and a deletion date which indicates the time at which that particular instantiation is no longer required. Note that adding a deletion date does not cause the instantiation to be removed; it marks it for removal at such time as the database containing it is 'cleaned up'. The provision of deletion date is useful in the construction phase of a project in providing an audit trail for additions and deletions which can then be costed in conjunction with the Cost model.

5.4.3.4 References

ISO 10303-WD106 (version S511);, Industrial automation systems and integration - Product data representation and exchange - Part 106: Building Construction Core Model

5.5 IFCPropertyTypeResource

5.5.1 PropertySets

5.5.1.1 Introduction

One of the primary issues with defining an object model for AEC projects is that there are literally thousands of building elements and concepts in these projects that are candidates to be represented as different types of objects. IFC Release 2.0 is focused on establishing a means by which information is shared and exchanged and does not specify any application behavior. This means that many of the distinctions between these objects can be defined in terms of ‘type’ and associated properties. Such PropertySets are associated with an object rather than contained in it.

This approach has two advantages:

- Dramatic reduction in the number of classes in an implementation - differences are captured in terms of attached PropertySets
- Opportunity to attach and detach such definition extensions at any time - thus an element can accumulate a richer and richer definition through time, as different application add PropertySets which define aspects from a particular application or AEC domain 'point of view'.

This will not work in the case where special semantic relationships are required for a particular ‘type’ of object. In these cases, 'types' must be promoted to object classes so that such special relationships can be modeled.

PropertySets are included in IFC Release 2.0 can be used in four ways:

1. Definition of Type driven properties to be shared by all occurrences of a given object type
2. Definition of Type driven properties that differ for each occurrence of a given object type.
3. 'Domain view' extensions to defined by classes in the IFC Core and Interoperability layers
4. Application and end-user defined extensions to objects defined by classes in IFC Core, Interoperability or Domain/Applications layers of the IFC object model.

The first two will be discussed in more detail in the section below on Type Definition. The latter two will be elaborated here.

5.5.1.2 Domain Extension PropertySets

It is anticipated that the most common use of PropertySets will be the extension of object definitions from the point of view of an industry domain, or software application. As IFC project models are to be shared between many disciplines, it is important to insure the availability of information that is commonly useful, without imposing the overhead that would be incurred by including all information of interest to all disciplines. Therefore, the IFC project model architecture is segmented. The IFC Core model includes classes that are needed by multiple disciplines. Their definition in the Core includes the information that is commonly useful. All other information about these objects is linked via extension PropertySets that are attached at any time in the project lifecycle, by any application. Further, as a project moves through its lifecycle, these extensions can be detached and archived when they are no longer needed online. Restoration from an archive and re-attachment should also be possible. In this way, any IFC compatible application has access to a wide array of "views" for any object in the project model, but does not have to deal with this extension information when it is not needed.

A number of Domain extension PropertySets are included in IFC Release 2.0. Applications supporting a particular domain extension model will be expected to be able to generate and correctly interpret these PropertySets.

EXAMPLE: Att_HVACSpaceElementInformation

*((** this section will be completed for the R2.0 final specifications **))*

5.5.1.3 Application/End-User Extension PropertySets

Unlike their use in type definition and Domain extensions, which are intended for use by multiple applications and users, End-User extensions will essentially be private to the application or end-user. Such private extensions are an important element in the IFC Project Model architecture because they insure the opportunity for applications and end users to extend and customize project model definitions on a project by project basis. This is an essential requirement for the AEC industry.

An application or end-user defined PropertySet can be developed at runtime. The IfcPropertySet and IfcProperty classes include the necessary attributes to create a custom set at any time. The IfcPropertySet class also includes a general purpose interface for querying the names and types of Properties included in it. This means that other applications will also have access to this information, though they may not correctly interpret the intended semanic meaning.

EXAMPLE: Att_MySpaceDefinitionExtension

*((** this section will be completed for the R2.0 final specifications **))*

5.5.2 TypeDefinitions

5.5.2.1 Introduction

This section defines the approach used in the IFC Object Model to defined types of objects that do not require separate object classes. This topic is closely related to the type driven PropertySets described in sections 5.5.2.3 and 5.5.2.4.

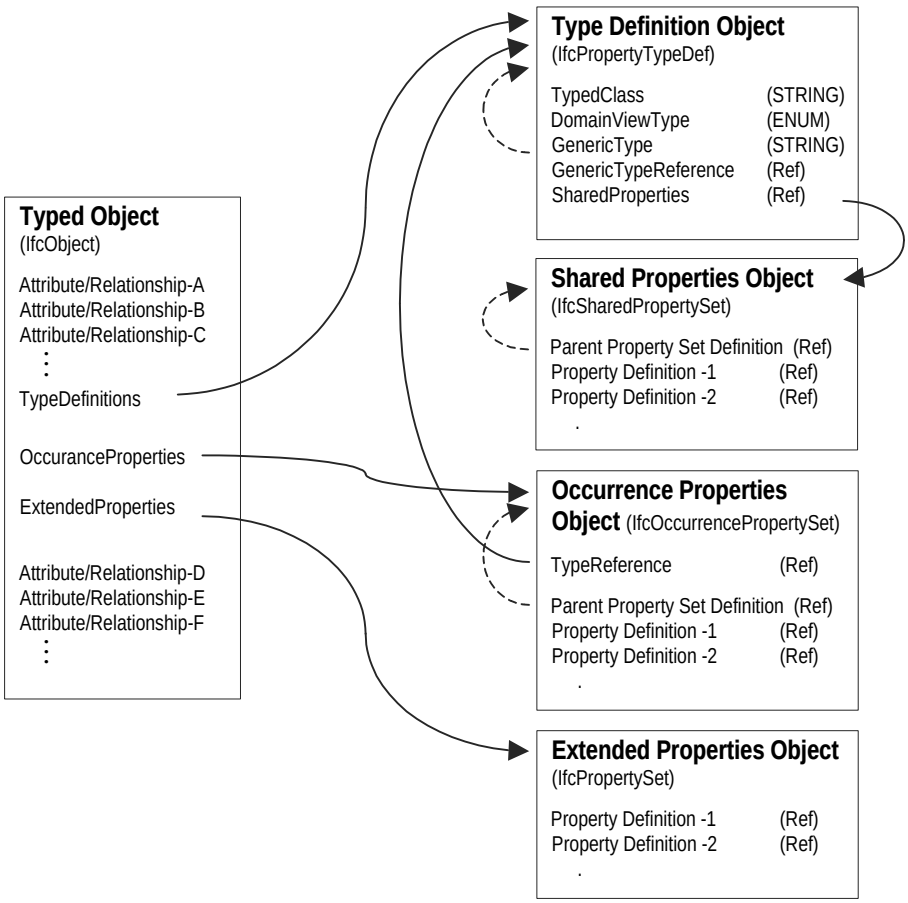
5.5.2.2 Runtime defined type definition

As IFC object 'Types' do not require separate classes for each type, they can be defined at runtime. This yields a significant advantage for software developers and end users of applications supporting IFC by providing flexibility and extensibility to the objects in IFC models.

5.5.2.3 Type Driven Attributes that are Shared

One motivation for defining a "Type" of building element is to establish a standard that will be used many times in a project. In these cases, a standard type is established through the definition of a set of attributes or characteristics that will be held constant for all occurrences in the Project model. Therefore, there should be a single record of these attributes that is associated with the TypeDefinition object rather than with each occurrence of the element.

Note that the TypeDefinition Class includes a relationship to an PropertySet for just this purpose.



EXAMPLE: Att_DoorType

*((** this section will be completed for the R2.0 final specifications **))*

5.5.2.4 Type Driven Attributes that vary with Occurrence

Another motivation for defining a “Type” of building Element is to establish a use or purpose for the element that requires a that a standard set of attributes be defined for each occurrence. In these cases, this standard set of attributes will be determined by the type, but values for these attributes will vary for each occurrence of the element type. Therefore, the must be a link from each occurrence of the element type to an “owned” occurrence of the PropertySet. Please see the diagram in the section above.

Note that the we have included a set of references to PropertySets in the IfcProductObject (called ExtensionPropertySets). In the case of a type driven PropertySet which varies for occurrence, the name of the PropertySet will be defined by the TypeDefinition object. An application must create an instance of this PropertySet and attach it as an ExtensionPropertySet on the typed object.

EXAMPLE: Att_OccupiedSpace

*((** this section will be completed for the R2.0 final specifications **))*

6. Guide to the Core Layer

*((** this section will be completed for the R2.0 final specifications **))*

7. Guide to the Interoperability Layer

*((** this section will be completed for the R2.0 final specifications **))*

8. Guide to the Domain/Application Models Layer

*((** this section will be completed for the R2.0 final specifications **))*