ifcXML

ifcXML language binding of EXPRESS

including ifcXML translation process
and ifcXML optimization process

Version 1.02
<table>
<thead>
<tr>
<th>Document Code</th>
<th>Title</th>
<th>Organization/Author</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG-01-001(Rev 6)</td>
<td>XML schema language binding of EXPRESS for ifcXML</td>
<td>Thomas Liebich</td>
<td>2001-12-11</td>
<td>second amendment</td>
</tr>
</tbody>
</table>

**Copyright © 2000-2001**

*International Alliance of Interoperability (IAI)*

*All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of the publisher (IAI).*
Acknowledgement

The work on ifcXML was initially contracted by the International Alliance for Interoperability, Inc., (North America) to Dr. Thomas Liebich to deliver a general mechanism to convert IFC EXPRESS definitions into IFC XML definitions.

Resources to further maintain, enhance and verify the ifcXML work and coordinate the results within ISO TC184/SC4 WG11 with the goal to achieve an international agreement of converting EXPRESS specifications into XML schema specifications had been provided by the IST project Intelligent Services Tools for Concurrent Engineering (ISTforCE), partially funded by the European Commission under IST-1999-11508.
# Table of Content

0 EXECUTIVE SUMMARY .................................................................................................................................................. 7  
  0.1 GOAL........................................................................................................................................................................... 7  
  0.2 DELIVERABLES............................................................................................................................................................... 7  

1 INTRODUCTION ................................................................................................................................................................. 8  
  1.1 SCOPE OF WORK .......................................................................................................................................................... 9  
  1.2 PURPOSE OF THE EXTRAPOLATION .......................................................................................................................... 10  
  1.3 AUTOMATIC XML TRANSLATION............................................................................................................................... 11  
  1.4 LIMITATIONS OF AUTOMATIC XML TRANSLATIONS............................................................................................ 11  
    1.4.1 Structural optimization ............................................................................................................................................... 12  
  1.5 CONFIGURATION OF BINDING..................................................................................................................................... 13  
  1.6 OTHER CONSIDERATIONS............................................................................................................................................... 13  

2 COMPARISON OF EXISTING XML LANGUAGE BINDINGS ....................................................................................... 14  
  2.1 CRITERIA FOR THE COMPARISON.............................................................................................................................. 14  
  2.2 COMPARISON OF THE AVAILABLE LANGUAGE BINDINGS.................................................................................... 15  
    2.2.1 Late bound markup declarations (LB) .......................................................................................................................... 15  
    2.2.2 EXPRESS-typed Early Binding (ETEB) ....................................................................................................................... 16  
    2.2.3 Object Serialization Early Binding (OSEB) .................................................................................................................. 16  
    2.2.4 BLIS-XML methodology for transforming EXPRESS data models to XDR .......................................................... 17  
    2.2.5 PNet XML schema approach for the representation of EXPRESS....................................................................... 18  
    2.2.6 XML schema for CEB (STEP tools) .......................................................................................................................... 19  
    2.2.7 Schema Language Optimized Binding (NIST) ........................................................................................................... 20  
  2.3 CONCLUSIONS ............................................................................................................................................................... 20  

3 XML SCHEMA LANGUAGE BINDING OF EXPRESS.................................................................................................... 22  
  3.1 FUNDAMENTAL CONCEPTS AND ASSUMPTION ....................................................................................................... 22  
    3.1.1 General requirements for the language binding ....................................................................................................... 22  
    3.1.2 General conditions for the EXPRESS source .......................................................................................................... 23  
    3.1.3 Mapped EXPRESS language declarations ............................................................................................................... 23  
    3.1.4 Unmapped EXPRESS language declarations .......................................................................................................... 24  
    3.1.5 Mapping of EXPRESS identifiers.......................................................................................................................... 24  
    3.1.6 Name spaces ............................................................................................................................................................ 24  
  3.2 MAPPING OF EXPRESS SCHEMA ................................................................................................................................. 25  
    3.2.1 Root element ........................................................................................................................................................... 26  
    3.2.2 Configuration attributes ........................................................................................................................................... 27  
  3.3 SIMPLE DATA TYPES ...................................................................................................................................................... 29  
    3.3.1 Simple data types in EXPRESS, SPF and XML schema, part 2 .............................................................................. 29  
    3.3.2 Mapping of EXPRESS simple data types .................................................................................................................. 30  
      3.3.2.1 Mapping of NUMBER data type ......................................................................................................................... 30  
      3.3.2.2 Mapping of REAL data type .................................................................................................................................. 31  
      3.3.2.3 Mapping of INTEGER data type ........................................................................................................................... 31  
      3.3.2.4 Mapping of BOOLEAN data type ........................................................................................................................... 31  
      3.3.2.5 Mapping of LOGICAL data type ........................................................................................................................... 31  
      3.3.2.6 Mapping of BINARY data type ........................................................................................................................... 31  
      3.3.2.7 Mapping of STRING data type ........................................................................................................................... 31  
      3.3.2.8 Mapping of Entity Instance Number .................................................................................................................... 32  
    3.3.3 Special XML built-in data types ............................................................................................................................... 33  
      3.3.3.1 Optimization for special built-in data types ........................................................................................................... 33  
      3.3.3.2 Configuration of the time declarations .................................................................................................................. 34  
  3.4 MAPPING OF NON-CONSTRUCTED DEFINED DATA TYPES ..................................................................................... 35
# Table of Figures

- **Figure 1**: Two stages of converting an EXPRESS model into XML ....................................................... 10
- **Figure 2**: Steps of the ifcXML generation process ............................................................................. 12
- **Figure 3**: Type and substitution hierarchy for element declarations ................................................. 55
- **Figure 4**: Definition of the IFCMeasureResource ............................................................................. 59
- **Figure 5**: XSD declarations of independent elements ........................................................................ 61
- **Figure 6**: XSD definition of IFCMeasureResource ............................................................................. 62
- **Figure 7**: IFC2x definition for SI unit ............................................................................................. 70
- **Figure 8**: ifcXML declaration for SI unit ......................................................................................... 70
- **Figure 9**: IFC2x definition for conversion based unit ......................................................................... 72
- **Figure 10**: ifcXML declaration for conversion based unit ............................................................... 73
- **Figure 11**: IFC2x definition of derived unit ...................................................................................... 75
- **Figure 12**: ifcXML declaration of derived unit ................................................................................ 76
0 Executive Summary

Currently the International Alliance for Interoperability has published its latest product, the IFC2x specification, which already receives major interests and implementation efforts. At the same time, several commercial and prototype implementations are available, that had been built according to the earlier IFC1.5.1 and IFC2.0 specifications. This work was and is done using the ISO/STEP based technology.

Meanwhile the XML based technology became commonly accepted and the mainstream commercial activity for many software and internet companies. As XML also allows to encode the content or the semantic of messages, XML is increasingly be used as technology for data exchange purposes. Acknowledging that, new initiatives, like aecXML in America and bcXML in Europe, have been established, which utilize XML as a technology for data exchange and sharing purposes.

However, one the major mistakes that had been made again, was the assumption, that XML also stands for the content of the message, and not only for the container of the message. The basic ideas, caveats, and time needs, that are inherent to any attempt to define (i.e. agree on) the content of the exchange (i.e. about the data professionals are using), also fully apply to XML based activities. Including the unavoidable problems occurring in any bigger and international organization, trying to achieve consensus on the content and structure of the exchange. Any change to leverage on existing standards, even if they come with some overhead, is beneficial to the new XML initiatives.

0.1 Goal

Therefore the goal of this project, the ifcXML extraction and evaluation project, is the provision of the internationally agreed content and structure of the IFC2x specification (and any valid subset thereof) to the XML community. The following use cases shall be acknowledged:

1. enable the exchange of IFC data files alternatively as XML instance documents
2. enable the reuse of IFC content and structure within XML based initiatives for data exchange and sharing in the construction and FM industries

The use cases reflect the combination of the requirements from the various interested parties, the IAI to offer more value to its membership by becoming interesting to the XML community, the BLIS group by considering the business case 1 and to the aecXML group by considering the business case 2. Technically the following have to be satisfied in order to support the use cases:

1. automatic translation of the IFC/EXPRESS source into ifcXML, which is governed by the ifcXML language binding of EXPRESS,
2. configuration of the ifcXML language binding to achieve optimized structures for the XML schema developers.

0.2 Deliverables

The ifcXML extraction and evaluation project achieves the goal by developing, discussing, and providing the following deliverables:

1. comparison of existing XML language bindings of EXPRESS
2. specification of the ifcXML language binding of EXPRESS
3. automatic translation of the IFC/EXPRESS source by a tool, written according to the ifcXML language binding with a stage 2 configuration to achieve optimized XML structures.
1 Introduction

The scope of the ifcXML extraction and evaluation project is the extrapolation of IFC definitions, currently available as EXPRESS data models (written according to ISO 10303-11:1994), to serve as schema specifications using XML technology (written according to the current W3C XML schema specification\(^1\)). The basic goal is the provision of an XML schema (XSD) specification to describe the structure of XML exchange data which is equivalent to the original information encoded in the IFC EXPRESS data model.

The XML specification is supposed to provide the content for the Common Object Schema Repository (COS/R\(^3\)), a repository of common definitions (in terms of XSD elements, attributes and types\(^4\)) that can be reused within other (more specialized) XML standards. By reusing the same set of common object definitions the more specialized XML standards can benefit from an area of interoperability among themselves (within the area of common reuse of the COS/R definitions).

The IFC model (within its current form, the IFC2x object model) comprises both, common object definitions and their reuse within several domain specific extensions. The common part of the IFC model is referred to as the IFC2x platform, the extensible part for special domain usage is referred to as the domain extensions. The IFC architecture governs the use of common parts and extension parts by applying a multi-schema layered architecture. Following the architectural rules the domain extensions share the same set of common platform definitions and thereby provide interoperability within this common part.

The goal of the ifcXML extraction and evaluation project is the provision of the common XML definitions coming from the IFC2x object model to serve as:

- the XML schema encoding of IFC information to be exchanged as XML document files alternatively to the existing method of STEP physical files. The mainly interested parties are software companies that want to support the exchange of IFC views both ways.
- the XML schema optimized encoding of relevant IFC information to be used as a common XML foundation or repository to build business case driven XML exchange specifications on top of it. The mainly interested parties are XML developers within initiatives, such as aecXML and bcXML.

Depending on the goals the following two levels of distribution are foreseen:

- The ifcXML of IFC2x – the scope of the ifcXML is the IFC2x object model, however as its minimum the IFC2x platform definitions.
- The ifcXML COS/R of IFC2x – the scope of the COS/R is the relevant subset of the IFC2x object model, however as its minimum the IFC2x platform definitions, which can be utilized as the vocabulary for a common object definition repository.\(^5\)

---

1 At the time of writing draft 1 until draft 3 of the ifcXML language binding, the latest XML schema specification was the Candidate Recommendation, published at 24. Oct. 2000. The specification is available at: [http://www.w3.org/TR/2000/CR-xmllschema-0-20001024/](http://www.w3.org/TR/2000/CR-xmllschema-0-20001024/)

2 At the time of submitting the final document on the ifcXML language binding, the XML schema specification had become an W3C technical recommendation, available at: [http://www.w3.org/TR/2001/REC-xmllschema-0-20010502/](http://www.w3.org/TR/2001/REC-xmllschema-0-20010502/). There had only been editorial changes compared with the Candidate Recommendation.

3 At the time of writing, there was uncertainty within the aecXML community on whether the objective is to utilize IFC definitions from either a common object schema (COS) or a common object repository (COR), therefore the combined term COS/R is used as a placeholder within this document.

4 The ifcXML specification is a collection of XML element, type, attribute, constraint and annotation definitions specifying the syntax and semantic of a structured content.

5 The exact nature of the COS/R is currently no clear and undergoes serious discussions within the aecXML community. Particularly it is unclear, whether the COS/R is a predefined subset (or view) of the information content provided by IFC2x, or whether is defines any (virtually possible) subset.
1.1 Scope of work

The ifcXML extraction and evaluation project aims at the goal to utilize the IFC specification, as the valuable asset of the IAI, in the XML language to foster its use by any, but particular web-based applications.

The basic approach will be as follows:

- Use of the existing IFC definitions, as defined in the latest IFC2x specification published by the IAI, written in EXPRESS to generate definitions optimized according to application and implementation requirements of XML technology and specifically according to the XML schema specification language.

- Review of the available formal XML language bindings, that have been developed so far by other organizations, or that are currently under development. These XML language bindings have been carried out using different versions of XML specification languages, mainly DTD\(^6\) and XML Schema\(^7\), in one case also XDR\(^8\).

- Specification of a formal language binding between the EXPRESS language definition and the XML schema language definition (in its current form). The XML schema language binding should enable:
  - Stage 1 translation – the automatic translation of the IFC object model from the EXPRESS language notation into the XML schema language notation, whereby the original EXPRESS syntax is fully maintained (100% 1 to 1 mapping) and all translation rules are driven by information from within the EXPRESS notation.
  - Stage 2 translation – the semi-automatic optimization of the outcome from the stage 1 translation to provide a more XML-alike language definitions. The configuration of the language binding should allow to describe and document the optimizations, whereas the following translation process should automatically execute the configured translation.

- Help on the definition of the nature and scope for the COS/R and indication of the relevant parts of the IFC2x object model that can be used within the specialized XML extension schemas.

The XML schema language binding of EXPRESS, as developed within the ifcXML extraction and evaluation project, will be used for an automatic translation of the whole IFC2x model (however as a minimum for the IFC2x platform model). The result will be the availability of the IFC2x model in XML schema notation. This deliverable is referred to as ifcXML for IFC2x and provides an alternative delivery form of IFC2x to the main delivery form using EXPRESS notation.

The ifcXML work on the XML schema language binding of EXPRESS will be submitted to the appropriate international body to standardize such work. It is the WG11 of the ISO (International Standardization Organization) TC184/SC4. This working group is currently collecting requirements for the second edition of Part 28\(^9\). Whereas the first edition is dealing with the XML 1.0 (DTD) language binding, edition 2 will focus on XML schema language binding.

To summarize: the ifcXML extraction and evaluation project will provide a methodology (the XML schema language binding) to translate the IFC2x model into XML schema and optimize the results, based on the evaluation or existing methodologies, and by providing input into the international standardization activities to secure the investments into ifcXML.

---

\(^6\) DTD is the Document Type Definition, provided as XML 1.0 recommendation of the W3C from 10. Feb 1998. The specification is available under http://www.w3.org/TR/1998/REC-xml-19980210.

\(^7\) See footnote 1.

\(^8\) XDR stands for XML Data Reduced.

1.2 Purpose of the Extrapolation

The term extrapolation has been used to combine the goals of automatic translation and following optimization within a single terminology. The two components are also classified as Stage 1 and Stage 2 translation steps and both follow the same basic methodology:

1. **Stage 1**: The automatic translation of the IFC2x EXPRESS model into an XML schema notation to facilitate the exchange of IFC data in XML format, as an alternative to the current way to exchange the IFC data in STEP physical file format. The translation preserves the exact syntax of the original EXPRESS model.

2. **Stage 2**: The optimized translation of the IFC2x EXPRESS model to serve as an XML object repository, which can be used to build XML exchange standards on top of it. The optimization is executed as a Stage 2 translation based on the results of the Stage 1 translation. In order to allow for optimized results, the Stage 2 translation process needs to be configurable. The configuration has to be informed by human interpretation of the underlying EXPRESS model to derive the most appropriate XML definitions. The optimization preserves the semantics of the original EXPRESS model, but may change the syntax to more appropriate XML structures. The resulting XML schema definition carries a signature of the configuration used.

Figure 1 shows the scenarios to apply a translation / extrapolation of existing EXPRESS model definitions into XML definitions. Whereas in Stage 1 the focus is on a (as close as possible) 1 to 1 mapping between the relevant data definitions within the EXPRESS source model and the XML target specification, Stage 2 requires an additional approach, as it includes the optimized mapping between potentially different data structures within the EXPRESS model and the XML specification.
There are two formal approaches possible to automatically translate between the XML schema definitions of stage 1 and stage 2.

1. To generate an intermediate EXPRESS models, optimized for the target optimized XML schema structure from the underlying IFC2x EXPRESS model using EXPRESS-X technology and map the resulting optimized EXPRESS model into XML schema definitions using the XML schema language binding to EXPRESS.

2. To generate the XML schema structure using the XML schema language binding to EXPRESS from the IFC2x EXPRESS model (stage 1) and then apply mappings to optimized XML schema definitions using e.g. XSLT translations (stage 2).

The performance criteria are also different depending on the stage chosen. In stage 1 the criteria are e.g. size of the resulting XML document exchange file, whereas in scenario 2 the criteria are more on the semantic richness of the resulting XSD definitions. In particular, an XSD definition, generated according to scenario 2, should be fully understandable for XML developers without referring back to the original EXPRESS definition.

1.3 Automatic XML Translation

The extrapolation shall be based on an automatic translation that shall be defined by an XML language binding of the EXPRESS language (or a mapping from EXPRESS to XML). The language binding shall define rules to translate all EXPRESS constructs for which counterparts exists in the XML schema Technical Recommendation into XML.

The IFC2x XML extraction project specifically acknowledges the work within the ISO TC184/SC4 Working Group 11 to add an EXPRESS language binding to XML schema to the upcoming version 2 of the international TS (Technical Specification) ISO-TS 10303-28. This work will be reviewed and considered as a foundation for the automatic XML schema translation.

1.4 Limitations of automatic XML Translations

Both languages, EXPRESS language and XML schema specification, have language constructs, which are not represented in the other language. Only those definitions, which are in the common set of both languages, can be translated.

An information model, like the IFC, have been developed with a certain set of criteria and with the capabilities of the specification language, EXPRESS, in mind. Therefore the IFC model contains many definitions, like the objectified relationship objects, the special entity data types for date and time, or the deep inheritance hierarchy, which are relevant under the criteria for the development of the IFC model, like modularization, cross discipline integration, or compliance with resources of the ISO standard 10303 – STEP.

This leads to a rather complex data structure within the IFC model, which, if directly translated, does not reflect the best usage of XML. The ifcXML extrapolation process therefore suggest the two processes

1. translation of the IFC EXPRESS source following the ifcXML language binding into XSD.

2. optimization of the XSD structure according to the choices, offered by the language binding, the choices may require semantic information, hidden in the explanations of the IFC model, to be unveiled by human interpretation.

Whereas the stage 1 can be fully automated, stage 2 will be semi-automated. Semi-automation refer to the human interaction to configure the language binding and the following automatic execution of that configuration.

Figure 2 shows the two components of the ifcXML extrapolation, the stage 1 automatic translation, and the stage 2 semi-automatic configuration of the optimized structure.
1.4.1 Structural optimization

One of the biggest problems when mapping an EXPRESS model into XML is the handling of entity references. EXPRESS does not distinguish between depending and depended entities (or classes) or between association (by reference) and aggregation (by containment). Using XML there is a distinction between a containment structure and a reference structure, whereas the containment structure is regarded to be the preferred and most XML-alike way to handle element relationships.

There is no information within the EXPRESS code that would allow to make a choice for either method, this information is only given in the explanations and interpretations of the EXPRESS model. There are four fundamental possibilities:

1. use only reference (it can deal with all cases, but it does not offer what is essential for XML),
2. use only containment (then duplicates of elements are needed in the exchange),
3. use always both, containment and reference,
4. use a notation in EXPRESS to guide the translator (to generate either reference or containment);

The most general solution (and which is chosen for stage 1 translation) is point 3 "use always both". The ifcXML language binding for stage 1 follows this principle to allow containment and reference for each EXPRESS entity reference translated\(^\text{10}\). The following assessments can be made:

1. does not utilizes the natural way how XML would preferably handle element relations,
2. is only useful for a query in a client/server use case, i.e. if an STEP database server gets a request from any web-based client to return a particular (small) part as XML data,
3. requires the provision of both cases (which causes a kind of overhead), but is most generic,
4. requires a manual interpretation of the original model and may lead to many uncertain situations (as the original model was not structured according to the containment/reference issue);

Within the stage 2 translation, the configuration of the translation allows a choice of either containment or reference structure (as proposed in points 1 and 2). Particularly point 2 (always containment) maybe

\(^{10}\) In comparison, the ISO 10303-28 language binding OSEB focuses totally on reference, whereas the proposed addition, the CEB (Containment Early Binding) focuses totally on containment.
useful, when translation particular subsets of the EXPRESS data model, where containment is always beneficial.

Allowing containment and references in a single XSD specification introduces an additional level of complexity (as two syntactically different concepts have to be foreseen for each child element being not a simple type). In order to handle this complexity, aspects of simplicity, such as proposed by the Simple Object Access Protocol, SOAP\textsuperscript{11}, should be taken into consideration, without attempting to be fully SOAP compliant.

Other potential structural improvements include (but are not limited to)

- Mapping of certain EXPRESS definitions, expressing data, time, duration, URI references and other information in some complex form into the simple XML built-in data types.

1.5 Configuration of binding

The configuration of the language binding to be executed in stage 2 provides a flexibility in applying the language binding for particular EXPRESS structures and semantics or for particular usages of the resulting XML definitions. Current configuration settings foresees the use for:

- mapping entity references into reference only, containment only or both (containment and reference)
- mapping STEP/EXPRESS\textsuperscript{12} time related definitions into XML using mapped elements, or using XML built-in types.
- applying meta tags to the XML schema definitions to retain more of the original EXPRESS model semantics. The particular implementation of this feature depends on future enhancements of the language binding.
- including the documentation of the originating EXPRESS model in the XML schema definitions using the documentation facet. The particular implementation of this feature depends on the documentation style used in the originating model.

1.6 Other considerations

The translation of the IFC2x object model into the ifcXML XSD structure shall be governed by the XML schema language binding and efforts shall be made to attract tool vendors to implement the language binding. In particular:

- The XML schema language binding to EXPRESS needs to be publicly available and shall have no restrictions coming from the IPR to its implementation,
- The IPR of the XML schema language binding should preferably belong to the IAI or to an official standardization committee, like ISO or W3C,
- Since there is a need for two application scenarios, to use the stage 1 translation to alternatively exchange IFC information in STEP physical file and XML file format, and to use stage 2 translation to base XML schema developments on top of a common repository, both translation shall be part of a single consistent language binding.

\textsuperscript{11} Currently a working draft, available at http://www.w3.org/TR/2001/WD-soap12-20010709/.
\textsuperscript{12} It handles the particular semantic definition of date_time_schema, as standardized in ISO 10303-41:1994.
2 Comparison of existing XML language bindings

Several organizations worldwide have already dealt with the development on XML language bindings to EXPRESS, most of the work was done either inside ISO TC184/SC4 or connected to this body. Within an IAI context, work was done by the BLIS group to develop a special language binding for its project companies. The efforts can first be grouped according to the XML notation which is supported.

The following efforts were known at the time of writing this document:

XML language bindings based on XML1.0 - DTD
- work published within the ISO/PDTR 10303-28 edition 1
  - Late bound markup declarations for EXPRESS-defined data (LB)
  - EXPRESS-typed Early Binding (ETEB)
  - Object Serialization Early Binding (OSEB)

XML language bindings based on XDR
- work within the BLIS group
  - BLIS-XML methodology for transforming EXPRESS data models to XDR

XML language bindings based on XML schema
- work proposed as requirements for ISO/TR 10303-28 edition 2
  - XML schema approach for the representation of EXPRESS schemas and data (ProSTEP)
  - XML schema for CEB (STEP tools)
  - Schema Language Optimized Binding (NIST)
  - ifcXML schema language binding (IAI)

2.1 Criteria for the comparison

The following criteria are used, when making the comparison and evaluation of the available methods. First the distinction is made according to the XML notation supported by the language binding. Second, a short assessment is made about the major business case, for which the methodology had been optimized. Third, the language binding is reviewed in more detail. The following items are used for comparison:

1. use of XML attribute versus XML element content
2. use of global element/attribute versus local element/attribute definitions
3. allowance for either reference, containment, or both structures
4. preservation of the inheritance tree
5. type safety – i.e. control about the referenced element types
6. use of special value type data types

From an ifcXML requirements view point, the following assessments can be made. First the use of XML schema as the appropriate XML notation is mandatory, second the major business cases should include the export/import of XML documents according to the EXPRESS structure and the self-explanatory nature of the XML alike translation. Third the following are the requirements to the language details:
1. element content is preferred in the ifcXML approach, attribute content should be preserved for id and meta information (e.g. for a query/return scenario)

2. local element/attribute definitions are preferred in the ifcXML approach (as this only complies with the object oriented nature of IFC specifications, i.e. the attributes are defined within the context of its class, not globally)

3. reference and containment is preferred in the ifcXML approach (as outlined before, there is a strong need for both cases)

4. inheritance hierarchy should be preserved to allow the extensibility within other XML schemas at all levels of the EXPRESS hierarchy (not only at the leaf-note level)

5. the schema should provide guidance and control about the types of the referenced elements (and thereby comply to the nature of the strongly typed EXPRESS resource)

6. the data types should allow for numeric and Boolean types in addition to string types

2.2 Comparison of the available language bindings

The language bindings as defined above are now characterized according to the criteria. A summary is given about the appropriateness for ifcXML and which language constructs have been adopted from that binding.

2.2.1 Late bound markup declarations (LB)

The late bound markup declarations of EXPRESS-defined data provides with the most generic encoding of any EXPRESS structure. The XML elements essential encode the EXPRESS SDAI dictionary schema (i.e. XML elements for ENTITY, SUPERTYPE, etc.), whereas the information of the EXPRESS model (or the schema), is encoded in XML data.

A late bound XML markup declaration set is one that can be used in the same manner for any EXPRESS schema. It does not define any constructs that are specific to the schema.

The following small example demonstrates it (quoted from ISO-PDTR 10303-28). The very simple EXPRESS model is translated into an XML instance documents (which attribute value assumption):

```xml
SCHEMA my_schema;
  ENTITY an_entity;
    attr1 : STRING;
  END_ENTITY;
END_SCHEMA;

<schema_instance express_schema_name="my_schema" id="#id1">
  <entity_instance express_entity_name="an_entity" id="#id2">
    <attribute_instance express_attribute_name="attr1">
      <string_literal>an attr1 value</string_literal>
    </attribute_instance>
  </entity_instance>
</schema_instance>
```

In consequence there is no need for a DTD to be generated for a schema (also not for the IFC EXPRESS). However the use of a late bound structure is very limited. For the business case 1 the file size is simply too large and the access to the information within a toolbox always has to the interpreted (potentially slow), for business case 2 there is simply no XML definition to be used for extensions. Therefore the LB approach has not been considered any further.
2.2.2  EXPRESS-typed Early Binding (ETEB)

The ETEB approach uses XML 1.0 (DTD) to provide an early bound of a particular EXPRESS schema. The major goal of ETEB is to maintain all EXPRESS information within the XML instance document. It is expressed as "maintains a high degree of the typing" which means that the data types declared in the EXPRESS schema (e.g., entity data types, defined data types) are explicitly maintained to the greatest degree possible in the XML declaration set produced by this specification.

An early bound XML markup declaration set is based on the specific EXPRESS schema and embeds specific aspects, such as names or structures, from the EXPRESS schema in the markup declaration set.

The following small example demonstrates it (quoted from ISO-PDTR 10303-28). The very simple EXPRESS model from the previous example is translated into an XML instance documents (which attribute value assumption):

```xml
<My_schema-schema id="id1" express_schema_name="My_schema"
  express_schema_identifier="My_schema Edition 2">  
  <An_entity id="id2"> 
    <An_entity.attr1> 
      <string>an attr1 value</string> 
    </An_entity.attr1> 
  </An_entity> 
</My_schema-schema>
```

The language provides:
1. for element content for data and for attributes to carry additional "meta" information – mainly the "EXPRESS typing",
2. for EXPRESS attributes that are defined as global XML elements (by DTD language restriction). However, using the naming `<express_entity_name>+"."+<express_attribute_name>` the uniqueness of the attributes originally defined within an entity is preserved,
3. for reference only, however using the method to create an empty element, named `<express_entity_name>+"-ref"` to ensure the typing, when providing the ID/IDREF reference link,
4. for maintenance of the inheritance structure, which is expressed in XML as containment of the subtype elements (with their content) within the supertype element,
5. for type safety implicitly, as the data type is preserved by using the element-ref convention (see point 3)
6. not for value data types (by DTD language restriction), however the information is implicitly preserved by using literal definitions (like integer-literal, real-literal)

To summarize, many of the requirements from ETEB are actually the same requirements for ifcXML. However by using XSD instead of DTD, those requirements can be far more effectively achieved. Another similarity is the use of role name and data type name for the encoding.

Using ETEB, particularly the handling of inheritance by a containment structure leads to potentially large and deeply nested files, since many of the EXPRESS information has to be carried in the XML instance document, rather then in the DTD definitions.

2.2.3  Object Serialization Early Binding (OSEB)

The OSEB approach uses XML 1.0 (DTD) to provide an early bound of a particular EXPRESS schema. The major goal of OSEB is to streamline the exchange of XML instance documents to achieve a flat, size-optimized structure.
“Object serialization” entails both the process of streaming objects into data structures and the process of creating objects from such data structures. Object serialization can be used whenever objects are written to or read from flat files, relational database tables, network transfer buffers, and so forth.

The following small example demonstrates it (quoted from ISO-PDTR 10303-28). The very simple EXPRESS model from the previous example is translated into an XML instance documents (which attribute value assumption):

```xml
<uos c="id2" schema="My_schema" unset="unset">
  <An_entity x-id="id2" Attr1-r="id3"/>
  <string x-id="id3">an attr1 value</string>
</uos>
```

The language provides:
1. for attribute content for data and idref(s)
2. for EXPRESS attributes translated into local XML attribute definitions
3. for reference only
4. for no maintenance of the inheritance structure, the abstract entities are not translated and the leaf node entities are flattened (all inherited attributes are explicitly declared at the instantiable leaf node)
5. no type safety is provided by the generated DTD
6. not for value data types (by DTD language restriction), however the information is implicitly preserved by using literal definitions (like integer-literal, real-literal)

To summarized, the OSEB only provides for the business case 1 (the exchange of EXPRESS models alternatively as XML documents). For the business case 2 (extensibility for new XML specifications) there are not sufficient information. In order to understand the OSEB encoding the original EXPRESS specification has to be used.

### 2.2.4 BLIS-XML methodology for transforming EXPRESS data models to XDR

The BLIS method uses XDR (XML data reduced) to provide an early bound of a particular EXPRESS schema. Although not explicitly stated, it is assumed that the EXPRESS schema has to comply to modeling rules similar to the IFC modeling rules (i.e. no inclusive subtypes, no multiple inheritance, etc.). The major goal of the BLIS method is to provide a flat and efficient XML exchange document structure to import/export IFC files alternatively as either P21 or XML files.

An example of BLIS-XML data.

```xml
<BLIS-XML xmlns="x-schema:BLIS_XML_FOR_IFC_2x.xml" schema="IFC2X">
  <IfcPropertySet XMLID="i2345" GlobalId="any_GUID" OwnerHistory="i1240" HasProperties="i1234 11235"/>
  <IfcPropertySingleValue XMLID="i1234" Name="any name" Description="any description">
    <NominalValue>
      <IfcValue>
        <IfcSimpleValue>
          <IfcLabel StringValue="number 1"/>
        </IfcSimpleValue>
      </IfcValue>
    </NominalValue>
  </IfcPropertySingleValue>
</BLIS-XML>
```
The BLIS method is described as straightforward and is strongly aligned to the EXPRESS based tool sets normally used. The following quotation demonstrates the design goals:

"BLIS-XML is a simple, straightforward methodology, based on consideration of what information software tools need from the source schema. The transformation is done such that data can be converted from Part21 files to BLIS-XML files and back to Part21 files without loss of information. This is possible, because the source EXPRESS schema is known and EXPRESS tools can map the information in BLIS-XML files to the full EXPRESS model using information in the source EXPRESS schema."

The language provides:

1. for mainly attribute content, i.e. most of the EXPRESS content is handled by XDR attributes, however there are exceptions in the methodology, e.g. for aggregates and select types, which are handled by element content. Therefore the method uses a mixture of attribute and element content.
2. for global definitions of attributes, i.e. all originally local EXPRESS definitions are transformed into global XML definitions. If this results into clashes (as the EXPRESS source defines attributes locally and attributes with same names does not need to have same types or semantics), the global definitions are locally overridden.
3. for only reference by using ID/IDREF(S) structures
4. for no preservation of inheritance, all inherited attributes are explicitly declared at the instantiate leaf node.
5. for no type safety, the language binding does not allow to verify whether the ID/IDREF(S) link is valid in terms of the data type of the linked XML element.
6. for value type data types, as supported by the simple data types of XDR.

To summarize, the BLIS method mainly provides for the business case 1 (the exchange of EXPRESS models alternatively as XML documents). It had been optimized to satisfy this case. For the business case 2 (extensibility for new XML specifications) there are many missing information, e.g. the inheritance structure, local content and type safety. In order to understand the BLIS-XML encoding the original EXPRESS specification has to be used.

2.2.5 PDnet XML schema approach for the representation of EXPRESS

The PDnet method uses XML schema for an early bound of a particular EXPRESS schema. The EXPRESS schema is subjected to some limitations, e.g. no multiple inheritance and no inclusive subtypes in order to allow for effective use of the language binding, although special rules within the language binding also allow to handle generic EXPRESS type inheritance.

The design goals for the PDnet method has been to achieve small file sizes for the large data sets normally used in the automotive industry. Some example file extracts:

```xml
<Plant xsi:type="Flower_plant" x-id="p:1" Name="lilly"
    Flower_colour="white"/>
</!---->
<Plant x-id="p:1" Colour="red" Latin_name="lilia"
    English_names="pne1 pne2"
    Survival_temperature_range="str1" />
<Plant_name-element x-id="pne1" value="lilly" />
<Plant_name-element x-id="pne2" value="red lilly" />
```

The language provides:

1. for attribute content for all EXPRESS attributes, with special treatment of string based aggregates, which have to be handled as entity references,
2. for local definitions of all attributes,
3. for reference by using ID/IDREF structures as preferred method, but also allowing alternatively containment – a special rule suggests reverse containment, i.e. that if the direct relationship is 1:1 and the inverse relationship is 1:N, then the containment shall be defined for the inverse relationship,
4. for preservation of the inheritance by type hierarchy (restricted to the single inheritance within the EXPRESS file),
5. for type safety by using key/keyref structures for all ID/IDREF combinations,
6. for value data types as supported by the XSD simple data types

To summarize, the PDnet approach uses the additional means provided by XSD to maintain a high degree of the typing present in the EXPRESS schema. The business case supported is the sending and receiving of large data sets, therefore file size become a critical issue. Due to this constraint some of the XSD definitions appear more cryptic than they would be expected from XML literates, such as the aggregates of string data types.

2.2.6 XML schema for CEB (STEP tools)

CEB uses XML schema for an early bound of a particular EXPRESS schema. The CEB defines a default mapping and alternate or refined mappings that may be designated through the use of CEB configuration tags.

The business case for CEB is the client/server scenario, where a web-based (and lean) client (e.g. a DOM based application) sends requests to a STEP based server and expects a return, which includes (normally small or medium-sized) data sets carried in an XML notation. Therefore the CEB method not only describes the language binding of the EXPRESS schema, but also some form of standard messages between the client and the server.

The following example shows the default containment mapping.

```xml
<E ceb:id = "I-30">
  <A_rel>
    <C ceb:id = "I-10">
      <a>1</a>
      <b>2.4</b>
      <c>hello</c>
    </C>
    <B_rel>
      <C ceb:id = "I-20">
        <a>2</a>
        <b>1.6</b>
        <c>goodbye</c>
      </C>
    </B_rel>
  </A_rel>
</E>
```

The language provides:
1. for element content only for all EXPRESS attributes, attribute content is preserved for id's and for the standard messages
2. for local definitions of sub elements (for the element content), and global definitions for associations, which are however unique due to the naming convention of `<express_entity_name> + "." + <express_attribute_name>
3. for reference and containment (driven by the configuration tags), in addition the so called "duplicate" tag allows the creation of exact duplicates of elements, when contained in more than one container
4. for no preservation of inheritance – the option was chooses to not have two different encodings for single and for multiple inheritance / inclusive subtypes.

5. for type safety by containment (current document does not describe the handling of the reference case, nor the use of key/keyref structures)

6. for value data types as supported by the XSD simple data types

To summarize, the CEB approach uses additional means provided by XSD to maintain a high degree of the original syntax to validate the derived XML document against the XSD schema. The configuration tags are an interesting addition to the language. The business case supported, i.e. the exchange of small portions or subsets of the original EXPRESS model population seems to be an appropriate use case combining the advantages of both languages.

2.2.7 Schema Language Optimized Binding (NIST)

The NIST XML schema language optimized binding has only recently been proposed. It uses (by own declaration) XML schema for an early bound of a particular EXPRESS schema. However the only available source was a TREX grammar notation. According to the author it is resembles many design choices within the ifcXML approach and provides:

1. for element content for all EXPRESS information (without id’s)
2. for local definitions
3. for reference and containment structures to be used interchangeable
4. not known
5. for type safety by using an empty element with IDREF
6. for values data types as supported by the XSD simple data types

To summarize, there is currently not enough information available to provide a more detailed assessment of the NIST approach. Therefore no suggestion is made on the usability of this method with the ifcXML context.

2.3 Conclusions

Depending on the supported use cases, different design choices have been made. If the use case it solely to exchange EXPRESS-based information alternatively to Part 21 (STEP physical file) also as XML documents, then it is often assumed, that the validation checks are done inside the EXPRESS components. In such use cases, often file size is used as a performance criterion. This leads to attribute content.

Other use cases include the exchange of XML data to be validated in XML components and to be readable by XML literates. Whereas in DTD mappings the additional syntax information had to be sent in the XML document, new XML schema based mappings allow to handle the validations within the schema definitions. In such cases, the extensibility and safety of the definitions is important and often element content and constraint components (key/keyref, etc.) are used.

Most XML schema mappings request a flexible handling of reference and containment to accommodate the advantages of XML containment and the flexibility of reference. Most XML schema mappings mandate the use of key/keyref structures to provide type safety. Different solutions are proposed for the more complicated constructs, such as selects, in the various approaches.

To summarize, the requirements for the ifcXML approach addresses the advantages of the various mappings always compared with both use cases of the ifcXML approach (see section 0), which are first the use of XML to exchange between two IFC/EXPRESS aware applications and second to provide an XML repository of IFC definitions for the XML development community.
Some of the other language bindings, most remarkably OSEB and BLIS, only address the first use case – which is explained in the relevant documents. OSEB particularly speaks about "serialization", and BLIS states the fact that "the logic of the EXPRESS model must be implemented, as it has been in components that load/use the EXPRESS schema".

The ETEB approach tries to keep as much as possible of the original typing within the XML structure, but due to the deficiencies of the DTD approach used, this leads to a considerable overhead of the XML exchange files.

The new XML schema based approaches PDnet and CEB try to overcome these restriction by encoding much of the original typing within the schema (and not the exchange file), so that XML aware validators can verify XML documents against these definitions. The CEB approach, offering configurations for the language binding, seems to be most interesting within the ifcXML context.
3 XML schema language binding of EXPRESS

The following draft of an XML schema language binding sets the basic requirements for an automatic translation of an EXPRESS model into XML schema (Technical Recommendation dated 2. May 2001\textsuperscript{13}). On the availability of other language binding specifications results of such effort are considered as additional input\textsuperscript{14}.

This part of the document specifies a mapping from an EXPRESS schema to an early bound XML markup declaration set called the ifcXML schema language binding. The language binding is applicable to every EXPRESS schema, within some limitations to the use of EXPRESS language constructs.

\textbf{NOTE} The limitations refer to those parts of the EXPRESS language which cannot be translated without a high degree of indirection. Many of those limitations are identical to the limitations of the use of the EXPRESS language constructs imposed by the IFC architecture and modeling guidelines.

The language binding is designed to be generally applicable to all EXPRESS defined schemas (within the detailed restrictions as mentioned below). Some additional guidance is given to the encoding of the IFC EXPRESS model, dealing with the particular modeling style used within IFC. Those additional recommendations are clearly marked within the document and they are not mandatory to the use of the language binding outside the International Alliance for Interoperability\textsuperscript{15}.

\textbf{NOTE} Example of clauses that are specific to the IFC model are the handling of the entity naming convention (particularly the Ifc prefix), the MixedCaseNamingStyle (in contrary to the underscore_naming_style) and the name space definition.

3.1 Fundamental concepts and assumption

The creation of the XML schema by applying the ifcXML language binding to the EXPRESS schema assumes a syntactically correct EXPRESS schema. To ensure the syntactical correctness the EXPRESS schema should pass an industry-strength EXPRESS parser.

The ifcXML language binding tries to maintain a high degree of the typing present in the EXPRESS schema. The typing information should preferably resist within the XML schema definitions and not within the XML instance document, so that an XML instance document can be validated according to the XML schema without the overhead to always carrying the information in the exchange file.

3.1.1 General requirements for the language binding

The following design goals govern the detailed development of the ifcXML language binding. These requirements are stated as:

- XML element content shall be used for data (referring to the EXPRESS attributes)

\textsuperscript{13} The XML Schema Technical Recommendation is available at http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/ (the XML Schema Part 0: Primer document).

\textsuperscript{14} At the time of writing the only available language binding document for XML schema was the "PDTnet approach for the XML representation of EXPRESS schemas and data", issued by the ProSTEP consortium, and the XML schema CEB binding from STEPtools. There is also a new proposal from NIST, for which however no further specifications have been available yet.

\textsuperscript{15} It is the attempt of the ifcXML language binding to not be restricted to the IFC EXPRESS model. The requirements of the ifcXML approach have been forwarded to ISO TC184/SC4 WG11 to be included in the Part 28 second edition.
XML element content shall be defined locally (similar to the EXPRESS original)
XML attribute content shall be used for id's, idref's and meta information (and remain extensible)
XML element content for entity relationships shall allow for containment and reference of the entity
  - Note: This may be restricted to either containment or reference by using a configuration to the language binding
XML element and type structure shall preserve the entity inheritance structure by substitution groups and type hierarchy
  - Note: This is restricted to single inheritance
XML attribute content shall be used for id's, idref's and meta information (and remain extensible)
XML element content for entity relationships shall allow for containment and reference of the entity
XML schema shall be "type safe" by using key/keyref constraint to control reference data type
XML elements shall use effective structure for aggregates of value data types
XML shall handle local and global id's of entities
the resulting XML schema shall be "XPath friendly"

3.1.2 General conditions for the EXPRESS source

The precondition to the EXPRESS schema used to create the XML schema declarations is its syntactical correctness. If the XML schema is generated from any subset of the EXPRESS schema, than this subset has to be fully self-contained and needs to be syntactical correct.

IAI-NOTE This applies to the view definitions and the platform subset of IFC2x as well as to the conformance classes of other EXPRESS based standards, such as ISO 10303

Some restrictions are applied to the EXPRESS source in order to be fully translated into the XML schema notation. The mapping beyond these restriction is out of scope for the ifcXML schema language binding. However it is expected that answers are given within the upcoming ISO/TR 10303-28 2nd edition. The restrictions are:

- inheritance – only single inheritance is supported, multiple inheritance is out of scope
- complex types – only exclusive (ONEOF) subtypes are supported, inclusive subtypes (AND, ANDOR) are out of scope
- multidimensional aggregates – only one-dimensional arrays, sets, lists and bags are supported.

IAI-NOTE The limitations ease the mapping into XML schema and allow to achieve the goal to "maintain a high degree of typing presented in the EXPRESS schema". Fortunately the restrictions are identical with the use of EXPRESS within the IFC Models, therefore the IFC EXPRESS definitions can be translated without restrictions.

The current version of the ifcXML is restricted to the translation of longform EXPRESS models, however future versions may also address short form EXPRESS models.

NOTE Under this restriction the shortform to longform translation needs to be carried out on shortform models before executing the language binding.

3.1.3 Mapped EXPRESS language declarations

The ifcXML language binding describes the mapping of the following EXPRESS declarations:

- Defined type, including
  - SELECT type
  - ENUMERATION type
- ENTITY type, including
  - SUBTYPE/SUPERTYPE (restricted to single inheritance)
### 3.1.4 Unmapped EXPRESS language declarations

The ifcXML language binding does not describe the mapping of the following EXPRESS declarations:

- **RULE**
- **FUNCTION**
- **PROCEDURE**
- **CONSTANT**
  - within the **ENTITY** declaration
    - **WHERE** rule (with some exceptions)
    - any other inheritance structure (beside single inheritance)
  - within the **TYPE** declaration
    - **WHERE** rule (with some exceptions)
  - **SCHEMA** interfacing
  - **REFERENCE** and **USE** statements

### 3.1.5 Mapping of EXPRESS identifiers

The following naming conventions shall apply to the EXPRESS identifiers (or entity, attribute, and type names) when translated into the XML names.

- **ENTITY** names shall be translated into XML **element** and **complexType** names whereby the XML names shall have the first character in upper case.
- **SELECT** type names shall be translated into XML **group** names whereby the XML names shall have the first character in upper case.
- **TYPE** names shall be translated into XML **simpleType** names whereby the XML names shall have the first character in upper case. This applies to all types (defined and enumerated) which are translated from the Express source.
- Attribute names (comprising attribute and relationship names) shall be translated into local XML **element** names with the first character in lower case.

An exception has to be made for EXPRESS identifier that begin with the characters "xml" regardless of case. These characters in the names shall be replaced by the character string "x-m-l".

**IAI-NOTE** Current IFC naming applies MixedCase for all entities, types and attributes. The MixedCase naming style shall be preserved (with first character being either lower or upper case). The naming convention applied shall follow the java capitalization scheme where classes (or entity versus element) are MixedCase and attributes and roles are camelCase.

### 3.1.6 Name spaces

Each EXPRESS **SCHEMA** shall define a name space. The name space identifier (a significant abbreviation of the schema name) shall be defined as **targetNamespace** of the XML schema specification as well. The namespace declarations for XML schema and XLINK shall be given as well.

```xml
<targetNamespace="<anyURI>"
xmlns="<anyURI>"
```
The name space URI should be a composition of the partial identifier denoting the EXPRESS to XSD binding method and the schema id. The name space may also denote the version date of the XSD specification release.

By providing the name space for all elements the name space abbreviation can be omitted. Therefore the value of the targetNamespace attribute of the xsd:schema element shall be a URI corresponding to the EXPRESS schema name. All definitions from the xsd namespace shall be fully qualified by the "xsd:" prefix.

IAI-NOTE The name space identifier for the IFC models shall be "ifc". The URI for the IFC models shall be specific to the IFC schema and the ifcXML encoding chosen. The URI for official versions of the ifcXML starts with "http://www.iai-international.org/ifcXML/". It should be extended with a schema specific name.

NOTE After final harmonization within ISO TC184/SC4 WG11, the URN for the international standard shall be used instead of the URI for the IAI based standard ifcXML. The URN shall then start with "urn:iso10303-28:xxx/ifcXML/", where xxx stands for the finally agreed abbreviation for Part 28 second edition.

IAI-NOTE The Ifc Prefix, which is applied to all IFC/EXPRESS ENTITY definitions shall be stripped from all translated XML names. The Ifc Prefix essentially indicates a name space, which shall be handled by the XML name space definition.

### 3.2 Mapping of EXPRESS schema

The EXPRESS longform `SCHEMA` shall be mapped into the XML schema "schema itself". The schema references the name space for the current version of the XSD definition and the target name space for the current IfcXML document.

```xml
<xsd:schema
targetNamespace
    = "http://www.iai-international.org/ifcXML/ifc2x_final"
xmlns = "http://www.iai-international.org/ifcXML/ifc2x_final"
xmlns:ifc2x = "http://www.iai-international.org/ifcXML/ifc2x_final"
xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
```

It is recommended to use qualified elements and unqualified attributes, however the language binding does not require a certain setting of this global or local attributes\(^\text{16}\). By default, the schema itself has:

```xml
| elementFormDefault = "qualified"
| attributeFormDefault = "unqualified"
```

Each EXPRESS `SCHEMA` also maps into a root element. By convention, this shall be the first `element` declaration contained within the schema itself. The root element contains all independent `element's`\(^\text{17}\) within the XML exchange file.

---

\(^{16}\) If compatibility with the SOAP is desired, it would demand to use unqualified elements and attributes.

\(^{17}\) Independent elements are also those elements, that can be referenced more than once.
The EXPRESS SCHEMA maps into a root element with the attribute name="ifcXML". The following example shows the conversion of the EXPRESS SCHEMA into XSD schema and root element.

```xml
SCHEMA IFC2X_FINAL;
...
END_SCHEMA;
```

In XML Schema:

```xml
<xsd:schema
  targetNamespace="http://www.iai-international.org/ifcXML/ifc2x"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:ifc2x="http://www.iai-international.org/ifcXML/ifc2x"
  xmlns = "http://www.iai-international.org/ifcXML/ifc2x">
  <xsd:element name="ifcXML">
    ...
  </xsd:element>
</xsd:schema>
```

In XML instance data:

```xml
<ifc2x:ifcXML
  xmlns:ifc2x="http://www.iai-international.org/ifcXML/ifc2x" ... >
```

**NOTE** After final harmonization within ISO TC184/SC4 WG11, the root element name for the international standard shall be used instead of the ifcXML name for the IAI based standard. The root name is likely to be "uos" (unit of serialization) for the finally agreed abbreviation for Part28 second edition.

### 3.2.1 Root element

The EXPRESS SCHEMA shall be mapped into the root element of the XML schema specification as described above. The root element contains a choice of element definitions, in which all non-abstract entity data types are included. In addition the root element contains all key and keyref definitions, which ensure the type safety for using element references by ID's.

The root element also declares its name, the name attribute value should be identical with the standard root name for the binding, currently name="ifcXML". In addition, the root element is further enhanced by attributes, which:

- optionally define the configuration tags, that had been used to guide the binding. The configuration tags are discussed in section 3.2.2.

The choice separator within the complexType declaration of the root element shall have a value maxOccurs="unbounded". The root element defines the document root node, in which all elements are contained and all the key and keyref constraints are defined.

```xml
<xsd:element name="ifcXML">
   <xsd:complexType>
     <xsd:choice maxOccurs="unbounded">
       <xsd:element ref="Classification" />
       <xsd:element ref="ClassificationNotation"/>
       <xsd:element ref="ClassificationNotationFacet"/>
       <xsd:element ref="ClassificationReference"/>
       <!-- all other non-abstract entities -->
     </xsd:choice>
     <xsd:attributeGroup ref="configTags"/>
     <xsd:attributeGroup ref="configTypes"/>
   </xsd:complexType>
   <!-- all key/keyref entries -->
</xsd:element>
```
3.2.2 Configuration attributes

For optimizing the resulting XML schema definitions, as discussed in section 1.4.1 the output can be configured within the stage 2 process. The configuration chosen to generate the XML schema definition needs to be visible within an XML document. Therefore additional attributes within the attributeGroup's for configurations are given for the root element.

Current configurations of the language bindings allow:

- the use of reference structures only, the use of containment structures only, or the use of a choice of containment or reference, for all entity references
- the explicit exchange of inverse attributes or omitting the exchange of inverse attributes
- the explicit exchange of derived attributes or omitting the exchange of derived attributes
- the replacement of ISO-10303-41 data type definitions by ISO 8601 date type definitions, as used directly by XML schema data types

Each root element shall therefore reference an attributeGroup definition with name "configTags", providing the following configuration tags:

```xml
<xsd:attributeGroup name="configTags">
  <xsd:attribute name="containmentType" default="choice">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="containmentOnly"/>
        <xsd:enumeration value="referenceOnly"/>
        <xsd:enumeration value="choice"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="inverseAttributes" default="exclude">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="include"/>
        <xsd:enumeration value="exclude"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="derivedAttributes" default="exclude">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="include"/>
        <xsd:enumeration value="exclude"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="timeDeclarations" default="iso10303">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="iso10303"/>
        <xsd:enumeration value="iso8601"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```
In addition the mapping of the simple data types, INTEGER, REAL, NUMBER, BINARY in EXPRESS into their counterparts in XSD may be subjected to a configuration.

```xml
<xsd:attributeGroup name="configTypes">
  
  <xsd:attribute name="integerType" default="long">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="short"/>
        <xsd:enumeration value="long"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="realType" default="double">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="decimal"/>
        <xsd:enumeration value="float"/>
        <xsd:enumeration value="double"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="numberType" default="decimal">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="decimal"/>
        <xsd:enumeration value="float"/>
        <xsd:enumeration value="double"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:attribute name="binaryType" default="hexBinary">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="hexBinary"/>
        <xsd:enumeration value="base64Binary"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

</xsd:attributeGroup>
```

The detailed mapping of simple datatypes is described within section 3.3.2 of this document. The default enumerator indicates the standard mapping.

A standard root element instance within an XML document (reflecting the stage 1 settings without any optimizations) would be:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ifcXML
 xmlns="http://www.iai-international.org/ifcXML/ifc2x_platform"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.iai-international.org/ifcXML/ifc2x_platform"
 containmentType="choice"
 inverseAttributes="exclude"
 derivedAttributes="exclude"
 timeDeclarations="iso10303"
 integerType="long"
 realType="float"
 numberType="float"
```
3.3 Simple data types

The built-in types, defined in Part 2 of XML schema, shall be preferably used. To facilitate such usage the built-in data types in this specification have the namespace URI:

http://www.w3.org/2001/XMLSchema

Since the ifcXML language binding specifically describes the usage of XML schema, the URI above is sufficient to address all built-in datatypes. The next clause compares the simple data types used in EXPRESS and XML.

3.3.1 Simple data types in EXPRESS, SPF and XML schema, part 2

The EXPRESS language defines the following simple data types:

- REAL, INTEGER, NUMBER, BOOLEAN, LOGICAL, BINARY, and STRING

REAL, INTEGER, BOOLEAN, BINARY, and STRING have the normal definition, the STRING as a sequence of characters as defined in ISO 10646 (Unicode). A NUMBER is the union of REAL and INTEGER and has in its domain all numeric values. The LOGICAL has as its domain the literals true, false, unknown. All implementation details, e.g. the bit length of a REAL, INTEGER or NUMBER, or the indication of signed versus unsigned INTEGER, are regarded as out of scope for EXPRESS.

The STEP physical file (as defined in ISO 10303-21 "Clear text encoding of the exchange structure") has the following data types to encode the simple data types in the EXPRESS language:

- integer, real, string, enumeration, binary and entity instance number

The integer is used to encode the INTEGER, the real is used to encode the REAL and NUMBER, the string is used to encode the STRING, the enumeration is used to encode the BOOLEAN and LOGICAL (and the EXPRESS constructed data type ENUMERATION), the binary is used to encode the BINARY and the entity instance number is used to define the reference to other ENTITY instances. Both forward and backward references are permitted.

The XML Schema definition language has a repertoire of built-in simple types. This is published as XML Schema Part 2: Datatypes. The basic (or primitive) XML data types are:

- string, boolean, decimal, float, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, QName, NOTATION

In addition, XML schema defines the following simple data types, which are derived by restriction from the primitive XML data types:

- normalizedString, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger

---

18 The URI for the usage of built-in datatypes outside of XML schema is: http://www.w3.org/2001/XMLSchema-datatypes.

19 The specification is available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
3.3.2 Mapping of EXPRESS simple data types

The EXPRESS simple data types, and the SPF entity instance number, shall be mapped into the following built-in data types used in XML schema. These built-in data types can then been used as types for attribute or element declarations corresponding to the EXPRESS attributes with underlying simple data types.

Example using simple data types for element declarations:

```xml
ENTITY IfcEntityWithSimpleTypes;
  aBinary : BINARY;
  aBoolean : BOOLEAN;
  anInteger : INTEGER;
  aLogical : LOGICAL;
  aNumber : NUMBER;
  aReal : REAL;
  aString : STRING;
END_ENTITY;
```

Resulting XML definition (only complexType declaration given here, for entity mapping see 3.6):

```xml
<xsd:complexType name="EntityWithSimpleTypes">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element type="xsd:hexBinary" name="aBinary"/>
        <xsd:element type="xsd:boolean" name="aBoolean"/>
        <xsd:element type="xsd:long" name="anInteger"/>
        <xsd:element type="logical" name="aLogical"/>
        <xsd:element type="xsd:decimal" name="aNumber"/>
        <xsd:element type="xsd:double" name="aReal"/>
        <xsd:element type="xsd:string" name="aString"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

And an example XML document part

```xml
<EntityWithSimpleTypes>
  <aBinary>10101010</aBinary>
  <aBoolean>true</aBoolean>
  <anInteger>1234</anInteger>
  <aLogical>unknown</aLogical>
  <aNumber>1234.5678</aNumber>
  <aReal>1.2892</aReal>
  <aString>Hallo</aString>
</EntityWithSimpleTypes>
```

The next headings explains each individual mapping of all EXPRESS simple data types.

3.3.2.1 Mapping of NUMBER data type

The NUMBER data type shall be represented by the `xsd:decimal` data type in XML schema by default. This mapping provides a means for a default representation of an unbounded, infinite precision numerical type for the minority of occasions in which it is a real requirement. It should be noted, that the NUMBER data type is rarely used in EXPRESS models.

The configuration attribute `name="numberType"` may force the mapping into the `double` or `float` type, if the default mapping is not appropriate. This is described in section 3.2.2.
3.3.2.2 Mapping of REAL data type

The REAL data type shall be represented by the xsd:double data type in XML schema by default. Although REAL in EXPRESS is unbounded real value (conceptually equal to xsd:decimal) a mapping to an IEEE double is required from an implementation point of view. See § 3.2.5 of the XML schema specification.

NOTE: If the double-precision 64-bit floating point type is not required to represent REAL datatypes throughout the schema, the customized use of xsd:float instead should be permitted.

The configuration attribute name="realType" may force the mapping into the decimal or float type, if the default mapping is not appropriate. This is described in section 3.2.2.

3.3.2.3 Mapping of INTEGER data type

The INTEGER data type shall be represented by the xsd:long data type in XML schema by default. The EXPRESS INTEGER represents an unbounded integral numeric type. Conceptually, the INTEGER type corresponds to the xsd:integer type; however, the default mapping in this binding is to xsd:long, also to align to the Java primitive types. See §3.3.13 of the XML schema specification.

The configuration attribute name="integerType" may force the mapping into the short or integer type, if the default mapping is not appropriate. This is described in section 3.2.2.

3.3.2.4 Mapping of BOOLEAN data type

The BOOLEAN data type shall be represented by the xsd:boolean data type in XML schema. See §3.2.2 of the XML schema specification.

3.3.2.5 Mapping of LOGICAL data type

There is no direct equivalence to the LOGICAL data type in XML schema. The XML simpleType corresponding to an EXPRESS data type LOGICAL shall be declared to have the attribute name="logical". This new simpleType has to be declared using the enumeration facet with the value="true", value="false" and value="unknown".

Resulting XML definition:

```xml
<simpleType name="logical">
  <xsd:restriction base="xsd:string">
    <enumeration value="true"/>
    <enumeration value="false"/>
    <enumeration value="unknown"/>
  </xsd:restriction>
</simpleType>
```

The simpleType "logical" shall be generated for each XML schema specification, where at least one LOGICAL type is used.

3.3.2.6 Mapping of BINARY data type

The BINARY data type shall be represented by the xsd:hexBinary data type in XML schema by default. See §3.2.15 of the specification.

NOTE XML schema primitive datatypes allow for two encodings of binary values, the "hex" encoding, where each binary octet is encoded as a character tuple, consisting of two hexadecimal digits, and the "base64" encoding, where the entire binary stream is encoded using the Base64 Content-Transfer-Encoding.
IAI-NOTE Since the IFC2x object model does not use the BINARY data type further discussion on which binary encoding should be used is out of scope of the ifcXML language binding.

The configuration attribute name="binaryType" may force the mapping into the base64Binary, if the default mapping is not appropriate. This is described in section 3.2.2.

3.3.2.7 Mapping of STRING data type

The STRING data type shall be represented by the xsd:string data type in XML schema. See §3.2.1 of the specification.

The STRING data type may be qualified in EXPRESS by a definition of its maximum width or its fixed width. The EXPRESS varying maximum width specification shall be translated into the XML constraining facet maxLength, the EXPRESS fixed width shall be translated into the XML constraining facet length.

Example

```
TYPE IfcGloballyUniqueId = STRING(22) FIXED;
END_TYPE;

TYPE IfcIdentifier = STRING(255);
END_TYPE;
```

Resulting XML schema definition:

```
<xsd:simpleType name="GloballyUniqueId">
  <xsd:restriction base="xsd:string">
    <xsd:length value="22"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Identifier">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="255"/>
  </xsd:restriction>
</xsd:simpleType>
```

Both the EXPRESS data type STRING and the XML data type xsd:string have at its domain a sequence of characters, the values of the characters are defined in ISO 10646 UNICODE.

3.3.2.8 Mapping of Entity Instance Number

The ENTITY instance number, which is used within the STEP physical file to exchange the entity instance references shall be translated into:

- ID for the entity instance number
- IDREF for the reference of an entity instance by number

The value for the EXPRESS entity instance number is guided by the following rule, it is a number sign, "#", followed by an unsigned integer. The ID and IDREF data type in XML shall follow the NCname production.

NOTE An ID value space is a name that has to begin with a letter or underscore (since "." has special meaning with XML namespaces, it is best to avoid using it). The initial character is followed by any number of letters, digits, periods, hyphens, underscores, or colons.

The value of the entity instance number should be translated into the value of type ID by replacing the first character "#" by ".".

Example

```
#100=IFCPERSON(/* attribute values */)
```
3.3.3 Special XML built-in data types

The XML schema part 2 built-in data types provide for more primitive and derived data types than EXPRESS. Several data types, such as the duration, dateTime (and the derived data types) and anyURI, have a particular meaning in the XML definitions, but can only be expressed as defined data types (or entity definitions) in EXPRESS.

Time, duration and URI references are also used in EXPRESS model, such as the IFC model. Since the semantics are contained in either the name identifier of the defined type or in the semantic description of the attribute or defined type the underlying base types, such as STRING or INTEGER can not be translated automatically into XML data types.

3.3.3.1 Optimization for special built-in data types

The following examples demonstrate the advantages of a guided and optimized translation for particular XML data types (here concentrating on the time relevant data types for date, time and data and time).

Example The following IFC definition of date and time (where rules removed), quoted from ISO 10303-41 date_time_schema (with the addition of the daylight saving offset) could be expressed by the following XML data types.

```
ENTITY IfcCalendarDate;
    DayComponent : IfcDayInMonthNumber;
    MonthComponent : IfcMonthInYearNumber;
    YearComponent : IfcYearNumber;
END_ENTITY;

ENTITY IfcLocalTime;
    HourComponent : IfcHourInDay;
    MinuteComponent : OPTIONAL IfcMinuteInHour;
    SecondComponent : OPTIONAL IfcSecondInMinute;
    Zone : OPTIONAL IfcCoordinatedUniversalTimeOffset;
    DaylightSavingOffset : OPTIONAL IfcDaylightSavingHour;
END_ENTITY;

ENTITY IfcCoordinatedUniversalTimeOffset;
    HourOffset : IfcHourInDay;
    MinuteOffset : OPTIONAL IfcMinuteInHour;
    Sense : IfcAheadOrBehind;
END_ENTITY;

ENTITY IfcDateAndTime;
    DateComponent : IfcCalendarDate;
    TimeComponent : IfcLocalTime;
END_ENTITY;
```

Using IfcDateAndTime as an underlying data type the base type would translate into xsd:dateTime, using IfcLocalTime as an underlying data type the base type would translate into xsd:time, and using IfcCalendarDate as an underlying data type the base type would translate into xsd:date.

Considering the following example

```
ENTITY IfcChangeOrder
    RequestedStartTime : OPTIONAL IfcDateAndTime;
    RequestedFinishTime : OPTIONAL IfcDateAndTime;
END_ENTITY;
```
Without optimization, it would lead to the following XML instance document

```xml
<ChangeOrder>
  <requestedStartTime>
    <DateAndTime id="1000">
      <dateComponent>
        <CalendarDate id="1010">
          <dayComponent>25</dayComponent>
          <monthComponent>01</monthComponent>
          <yearComponent>2001</yearComponent>
        </CalendarDate>
      </dateComponent>
      <timeComponent>
        <LocalTime id="1011">
          <hourComponent>10</hourComponent>
          <minuteComponent>25</minuteComponent>
          <zone>
            <CoordinatedUniversalTimeOffset id="_1012">
              <hourOffset>01</hourOffset>
              <sense>Ahead</sense>
            </CoordinatedUniversalTimeOffset>
          </zone>
        </LocalTime>
      </timeComponent>
    </DateAndTime>
  </requestedFinishTime>
</ChangeOrder>
```

Using the `xsd:dateTime` data type as the type for the `RequestedStartTime` and `RequestedFinishTime` elements, the exchange structure could be optimized.

With optimization, it would then lead to the following XML instance document

```xml
<ChangeOrder>
  <requestedStartTime>2001-01-25T10:25:00+01:00</requestedStartTime>
  <requestedFinishTime>2001-02-12T18:00:00+01:00</requestedFinishTime>
</ChangeOrder>
```

3.3.3.2 Configuration of the time declarations

If the configuration tag is set to `timeDeclarations="iso8601"`, then the creation of the XML element and complexType definitions for:
IfcDateAndTime
IfcLocalTime
IfcCalendarDate

should be skipped and the XML primitive datatypes shall be assigned directly to all EXPRESS attributes having the datatype of the above mentioned entities.

IAI-NOTE: The current IFC2x specification uses only these definitions from the date_time_schema of ISO10303-42, others, as ordinal_date and week_of_year_and_day_date, are not used. The optimized mapping of those additional date and time definitions found in ISO10303 are therefore out of scope of the ifcXML language binding.

3.4 Mapping of non-constructed defined data types

A non-constructed defined data type is an EXPRESS defined data type whose final underlying type is not a SELECT type or an ENUMERATION type. Each non-constructed defined data type shall be mapped into an XML simpleType declaration.

3.4.1 Mapping of non-aggregated, non-constructed defined data types

For each defined data type with (non-aggregated) simple base data type, the XML Schema Definition should contain a new simpleType declaration, derived by restriction from the corresponding built-in type, as defined in clause 3.3.2.

Considering the following EXPRESS definition:

```express
TYPE IfcLabel = STRING;
END_TYPE;
```

The XML schema definition should be:

```xml
<xsd:simpleType name="label">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

For each defined data type with a base type being a defined data type as well, the XML Schema Definition should contain a new simpleType declaration, derived by restriction from the other simpleType declaration.

Considering the following EXPRESS definition (IfcRatioMeasure corresponds to ISO 10303-41 ratio_measure):

```express
TYPE IfcRatioMeasure = REAL;
END_TYPE;

TYPE IfcNormalisedRatioMeasure = IfcRatioMeasure;
WHERE
    WR1 : {0.0 <= SELF <= 1.0};
END_TYPE;
```

The XML schema definition should be:

```xml
<xsd:simpleType name="ratioMeasure">
    <xsd:restriction base="xsd:double"/>
</xsd:simpleType>

<xsd:simpleType name="normalisedRatioMeasure">
    <xsd:restriction base="ratioMeasure"/>
</xsd:simpleType>
```
3.4.1.1 Optimization for subset of WHERE rule constraints on defined data types

An EXPRESS defined data type can have a domain rule, a WHERE clause, which restricts the domain of the defined data type. An XML simpleType can have constraining facets, which further constrain the domain of values. A subset of the possible ways to express a WHERE rule can be translated into constraining facets.

NOTE This language binding does not attempt to define automatic translation rules to map certain WHERE rule expressions into XML constraining facets.

The mapping of the previous example regarding the IfcNormalisedRatioMeasure could be optimized by using the constraining facets xsd:minInclusive and xsd:maxInclusive.

The XML schema definition should be:

```xml
<xsd:simpleType name="normalisedRatioMeasure">
  <xsd:restriction base="ratioMeasure">
    <xsd:minInclusive value="0.0"/>
    <xsd:maxInclusive value="1.0"/>
  </xsd:restriction>
</xsd:simpleType>
```

3.4.2 Mapping of aggregated, non-constructed defined data types

Aggregation data types in EXPRESS include the data types OF ARRAY, LIST, SET and BAG. Defined non-constructed data types can have an aggregated base data type.

For each defined type declarations with an aggregated base data type the XML schema definition shall contain a new simpleType with the list descriptor.

The aggregate bounds used in EXPRESS define the minimum and maximum number of elements that are valid for the aggregate. If the aggregate bound is given, and if it is not equal to [0:?] than it should be translated into the constraining facets minLength and maxLength.

minLength shall have the value of the lower bound, or it should be omitted if the lower bound evaluates to "0", maxLength shall have the value of the upper bound, or it should be omitted if the upper bound evaluates to "?".

Considering the following EXPRESS definition:

```latex
TYPE IfcCompoundPlaneAngleMeasure = LIST [3:3] OF INTEGER;
END_TYPE;
```

The XML schema definition should be:

```xml
<xsd:simpleType name="compoundPlaneAngleMeasure">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="xsd:long"/>
    </xsd:simpleType>
    <xsd:minLength value="3"/>
    <xsd:maxLength value="3"/>
  </xsd:restriction>
</xsd:simpleType>
```

There are several exceptions to the translation of aggregated defined data types into xsd:list type.

- An aggregate of STRING cannot be translated into xsd:list. The restriction is due to the fact, that the XML simple list type allows only lists of strings separated by white spaces, i.e. if
one of list elements would contain one or several white space, it would result into two (or more) list elements.

- An aggregate of aggregate cannot be translated into `xsd:list` as an XML list type cannot have a base type being a list.

*IAI-NOTE* Since the IFC2x object model does not include multi-dimensional aggregates, handling of this case is not further explored in this language binding document.

### 3.4.2.1 Mapping of aggregated STRING types

The mapping of types being an aggregate of STRING types should lead to the definition of a wrapper element having the `complexType`, resulting from the mapping of STRING aggregates. The creation of the wrapper element should follow the principle as outlined in 3.6.6.2.

### 3.5 Mapping of constructed defined data types

A constructed defined data type is an EXPRESS defined data type whose final underlying type is either a `SELECT` type or an `ENUMERATION` type. Each constructed defined data type being an `ENUMERATION` shall be mapped into an XML `simpleType` declaration. Each constructed defined data type being a `SELECT` shall be mapped into an XML `group` declaration.

#### 3.5.1 Mapping of Enumeration types

An `ENUMERATION` datatype has as its domain an ordered set of names. The names represent values of the `ENUMERATION` data types. See also ISO 10303-11 clause 8.4.1.

For each EXPRESS `ENUMERATION` data type, the XML Schema should contain a corresponding XML `simpleType` declaration derived from `xsd:string` type using the enumeration facet. Each enumeration item in the EXPRESS type definition should be represented as enumeration value.

Considering the following EXPRESS definition:

```express
TYPE IfcAddressTypeEnum = ENUMERATION OF
  (OFFICE,
   SITE,
   HOME,
   DISTRIBUTIONPOINT,
   USERDEFINED);
END_TYPE;
```

The XML schema definition should be:

```xml
<xsd:simpleType name="AddressTypeEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="office"/>
    <xsd:enumeration value="site"/>
    <xsd:enumeration value="home"/>
    <xsd:enumeration value="distributionpoint"/>
    <xsd:enumeration value="userdefined"/>
  </xsd:restriction>
</xsd:simpleType>
```

*IAI-NOTE* Enumeration values in EXPRESS are case insensitive and are normally given all in upper case. To comply with normal XML usage, enumeration values should be converted into all lower case for translation into XML schema.
3.5.2 Mapping of Select types

A SELECT data type in EPRESS has as its domain the union of the domains of the named data types within the select list. The SELECT data type is a generalization of each of the named data types in its select list. See also ISO 10303-11 clause 8.4.2. When used as the underlying data type of an ENTITY attribute, a single instance of a particular data type from the select list is created.

For each EXPRESS SELECT data type, the XML Schema should contain a corresponding XML group declaration of a model group\(^{20}\). Each select item in the EXPRESS type definition should be represented as an element within the choice compositor.

The SELECT data type can have in its select list ENTITY data types, constructed and non-constructed defined data types. The mapping of the elements within the choice compositor depends on the type of the select list member.

3.5.2.1 Mapping of select list members of type ENTITY

Each ENTITY typed SELECT list member should map into a sub element, referencing the global element declaration of the converted ENTITY datatype. It therefore references the element definition directly <xsd:element ref="..."/>

Considering the following EXPRESS definition (quoted from ISO 10303-42 axis2_placement):

```plaintext
TYPE IfcAxis2Placement = SELECT
( IfcAxis2Placement2D,
 IfcAxis2Placement3D );
END_TYPE;
```

The XML schema definition of the EXPRESS example given above should be (for the case of "containmentType" being "choice" or "referenceOnly"):

```xml
<xsd:group name="Axis2Placement">
<xsd:choice>
  <xsd:element ref="Axis2Placement2d"/>
  <xsd:element ref="Axis2Placement3d"/>
</xsd:choice>
</xsd:group>
```

**NOTE** By using the element declaration the XML instance document can later include the axis placements by containment within the entity which attribute data type is the SELECT. Then the element has no idref attribute, but contains all of its elements. But it can also include the axis placements by reference. Then the element has the idref attribute given and the attribute xsi:nil="true".

The XML document may now contain usages of the SELECT typed attribute being either containments or references. However by using the configuration tag "containmentType" it may be restricted to either "referenceOnly" or "containmentOnly". More information about the single and multiple references of element's is given within the section of entity mapping (see 3.6). The element declaration is qualified by nillable="true", and therefore allowing for empty embedded elements. However if the "containmentType" is "containmentOnly" then nillable="false" since no empty elements are permitted.

The XML document following the XML schema definition above should be for containment and reference of the ENTITY typed SELECT type:

---

\(^{20}\) The use of model group instead of complexType is preferred as it does not add to the type hierarchy – similar to EXPRESS, where SELECT does not add a type – it is rather a discriminated union of its select types.
3.5.2.2 Mapping of list members being defined data types or enumerations

If the SELECT list member is a non-constructed defined data type, or an enumeration data type, it should map into a sub element within the choice compositor. It thereby creates a wrapper element for the simpleType declaration, which is declared locally.

The corresponding simpleType shall be used as base types for the element declaration `<xsd:element name="..." type="..."/>`.

**Considering the following EXPRESS definition (quoted from ISO 10303-42 trimming_select):**

```express
TYPE IfcTrimmingSelect = SELECT
  ( IfcCartesianPoint,
    IfcParameterValue );
END_TYPE;

TYPE IfcParameterValue = REAL;
END_TYPE;
```

The XML schema definition should be:

```xml
<xsd:group name="TrimmingSelect">
  <xsd:choice>
    <xsd:element ref="CartesianPoint"/>
    <xsd:element name="ParameterValue"/>
  </xsd:choice>
</xsd:group>

<xsd:simpleType name="ParameterValue">
  <xsd:restriction base="xsd:double"/>
</xsd:simpleType>
```

See also clause 3.6.5 on the exposure of the role (or attribute) name and the data type. For SELECT it is similar to the exposure of the parameter name in ISO 10303-21. The indication of the data type is essential to re-establish the correct value.

3.5.2.3 Mapping of list members being of SELECT datatype

If the SELECT list member is a SELECT data type it should map into a sub GROUP within the choice compositor. It references the global group declarations (as translated from the SELECT definition) directly `<xsd:group ref="..."/>`.

**Considering the following EXPRESS definition:**
The XML schema definition should be:

```xml
<xsd:group name="Value">
  <xsd:choice>
    <xsd:group ref="MeasureValue"/>
    <xsd:group ref="SimpleValue"/>
    <xsd:group ref="DerivedMeasureValue"/>
  </xsd:choice>
</xsd:group>
```

IAI-NOTE There is a limitation, if the SELECT graph is not a tree. Since the IFC model only incorporates SELECT definitions with graphs as trees, no further attempt is made to solve this exception.

### 3.6 Mapping of ENTITY declarations

The ENTITY declaration defines an ENTITY data type, which includes attribute definitions that represent properties of the entity and may be associated with a value in each entity instance. The data type of the attributes establishes the domain of possible values. In addition an ENTITY declaration can have constraint definitions on attribute values and on relationships among attributes and other ENTITY instances. See also ISO 10303-11 clause 9.2.

Associations between entities are defined as attributes having the type of the referenced ENTITY as underlying data type. In an exchange file, ENTITY references are always handled by referring to the entity instance number, or by reference. Associations by value, where the references ENTITY instance is contained by the referencing ENTITY instance are not in scope of EXPRESS.

XML mandates the use of containment for element instances as the default option, but also provides the ID/IDREF mechanism for those cases, where containment is either not possible or feasible.

The XML schema language binding of the ENTITY data type should incorporate the ability to use entities as local element declarations within other element declarations by:

- containment
- reference

As expressed in section 1.5 and 3.2.2, the configuration of the language binding allows for either containment and reference, only containment or only reference on a global scale (i.e. applicable to the whole schema). Depending on the setting of the configuration tag "containmentType", there are different mapping and instantiation instructions on ENTITY declaration and attribute mappings.
By default both containment and reference of ENTITY type attributes should be supported. The method to provide both methods had been adapted from SOAP\textsuperscript{21}, where the principle is explained as "single-reference" versus "multi-reference".

SOAP §5.1-10: Given the information in the schema relative to which a graph of values is serialized, it is possible to determine that some values can only be related by a single instance of an accessor. For others, it is not possible to make this determination. If only one accessor can reference it, a value is considered "single-reference". If referenced by more than one, actually or potentially, it is "multi-reference."

Single-reference values (or containment of elements) can appear at any level within the document, whereas multi-reference values (or referenced elements) can only appear at the top level of the document node. Elements under the document node are referred to as "independent", whereas all other elements are considered to be "embedded".

SOAP §5.1-11: Syntactically, an element may be "independent" or "embedded." An independent element is any element appearing at the top level of a serialization. All others are embedded elements.

A multi-reference value is encoded as an independent element containing a local, unqualified attribute named "id" and of type xsd:ID. Each accessor to this value is an empty element having a local, unqualified attribute named "href" and of type xsd:IDREF, with a "href" attribute value referencing the corresponding independent element.

NOTE The ifcXML specification derivates from the original SOAP specification by making "href" of type xsd:IDREF, whereas SOAP has it as xsd:anyURI. The reason is to allow the use of xsd:key and xsd:keyref structures together with "id" and "href" attributes.

The accessor element (or referencing element) contains the attribute xsi:null with value "true". Therefore the element declaration needs to permit the use of xsi:null by the declaration attribute nullable="true".

### 3.6.1 ENTITY mapping and base element declaration

For each entity (which is not a subtype of another entity) the following mapping into the XML schema declarations element and corresponding complexType should apply.

The element declaration shall have the name attribute with a value equal to the ENTITY identifier. The element should have a type, provided by an xsd:complexType with the name attribute with a value equal to the ENTITY identifier. This pair should be generated for all ENTITY definition.

The xsd:complexType which corresponds to the ENTITY definition is derived from the base type of all element declaration (directly if the ENTITY does not have a SUBTYPE declaration, or indirectly if it has a SUBTYPE declaration). This base type shall be an xsd:complexType with the attribute value name="element". It defines two additional attributes

- xsd:attribute with name="id" and with type="xsd:ID".
- xsd:attribute with name="href" and with type="xsd:IDREF".

The first attribute "id" provides for the local id, which is unique within the XML document instance. It is required to be instantiated for all independent elements (at the top of the document node), and is also required for embedded elements, if they do not refer to a multi-reference value (i.e. for non-accessor elements). The second attribute "href" provides for a reference to an independent, multi-reference value within the same XML document. It has to be instantiated for all accessor elements.

\textsuperscript{21} Note that SOAP is still a working draft and may be subjected to changes. The current document is available at: http://www.w3.org/TR/2001/WD-soap12-20010709/.
NOTE: At no circumstances both, the "id" and the "href" should be given at the same element within the XML document. This cannot be enforced by XML schema definitions (as XML schema does not provide the same power of constraint definitions as EXPRESS), but need to be guaranteed by implementations.

IAI-NOTE: In addition the base element may also contain a third attribute xsd:attribute name="guid" with the type="xsd:anyURI". This attribute, which should be optional in usage, provides for the global id, that can be used to provide identities across various XML documents. However the provision of the GUID attribute is not required by the ifcXML.

The XML schema document shall have one occurrence of the complexType declaration for the base element.

```xml
<xsd:complexType name="element">
  <xsd:attribute name="id" type="xsd:ID" use="optional"/>
  <xsd:attribute name="href" type="xsd:IDREF" use="optional"/>
</xsd:complexType>
```

Each of the element declaration shall get these attributes by defining a complexContent and extending the base element type (see 3.6.3).

### 3.6.2 Configuration of the ENTITY declaration mapping

If the configuration tag "containmentType" is set to "choice" (the default), or "referenceOnly" no changes are made. If the configuration tag "containmentType" is set to "containmentOnly", then the corresponding base element type is a complexType with the attribute name="element" and only the optional attribute "id" given.

The XML schema document shall have one occurrence of this complexType declarations.

```xml
<xsd:complexType name="element">
  <xsd:attribute name="id" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

NOTE: The mandatory use of containment provides severe restriction on the structure of the schema definition, as sub elements cannot be shared (like instance references). One possibility to overcome the situation is to allow duplicates of element instances in the exchange document. This maybe considered in future versions of the language binding.

IAI-NOTE: Currently the use of "containmentOnly" is only advisable within subsets of the IFC model that demand containment as the most effective exchange structure.

### 3.6.3 Entity mapping by an element and type declaration.

To allow building a type hierarchy, the type declaration shall be done outside the content of the element declaration and the element declaration shall use the type attribute to refer to the element type.

The type of the element declaration shall be given by an xsd:complexType with the name attribute having the value of the ENTITY identifier. It should be derived by extension from the base type of all elements, the xsd:complexType with name="element" (normally the extension\(^{22}\) is used by adding the element and attribute definitions mapped from the EXPRESS attributes).

\(^{22}\) The use of restriction maybe discussed in future versions of the language binding, when the handling of re-declared attributes is revisited.
Considering the following EXPRESS definition:

```express
ENTITY IfcPerson;
-- all attributes omitted
END_ENTITY;
```

The default translation into XML schema is:

```xml
<xsd:element name="Person" type="person" nillable="true"/>
<xsd:complexType name="Person">
  <xsd:complexContent>
    <xsd:extension base="element"/>
  </xsd:complexContent>
</xsd:complexType>
```

### 3.6.4 Ensuring consistency between element and element reference

In addition to the conventions taken from the SOAP approach, i.e. the use of "id" and "href" attributes, the consistency and type safeness should be ensured by additional declarations of `xsd:key` and `xsd:keyref` components. This consistency check is demanded for references of attribute values (multi-references according to SOAP), and therefore necessary if the configuration tag "containmentType" is set to "choice" (the default value), or to "referenceOnly".

The mapping and instantiation of an entity type attribute may lead to two `element`’s within an XML instance document, the embedded empty accessor `element` with the "href" attribute and the independent `element` with the "id" attribute under the root node.

Within an XML instance document it shall be ensured that there is an instance of the independent `element`, if there are one of many instances of the embedded empty `element`, referring to that independent `element`. The equivalent XSD definition components are the `xsd:key` and the `xsd:keyref` components, each referring to the `key` and `keyref` by an XPath expression.

The use of the identity constraint definition schema components `key` and `keyref` shall be enforced to ensure the integrity mentioned above. The `key` and `keyref` definitions shall be inserted in the scope of the root element declaration (see also 3.2). The selector and field components shall have an XPath expression pointing to the independent `element` and embedded element reference.

Since independent element’s do only exists at the top of the document node, the `xsd:selector xpath` expression for `xsd:key` refers to the child of the document node (where it is defined) directly. It also ensures, that the field attribute "id" is given. That imposes, that for all independent elements the "id" attribute has to be given. The `xsd:selector xpath` expression for `xsd:keyref` refers to all descendants of the document node, for which the field attribute "href" is given. These are all embedded `element`s that are accessors to the independent `element`.

In order to support modularization (i.e. the later support of multi-schema – or short form – EXPRESS models), the `key` and `keyref` names shall be fully qualified, by using `SCHEMA` name + "," + `ENTITY` name.

Considering the following EXPRESS definition:

```express
SCHEMA Ifc2x_final;
ENTITY IfcOrganization;
-- all attributes omitted
```

---

23 The Xpath is defined in the XML Path Language (XPath), see http://www.w3.org/TR/xpath

24 The definition of `xsd:key` within http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ demands the existence of the field attribute. See § 3.11.4 Identity-constraint Definition Validation Rules.

25 In contrary to the `xsd:key` constraint, the field of `xsd:keyref` does not need to be given, it is only taken into account for those descendants for which it is available.
This should translate into the following XML schema definition for `xsd:key` and `xsd:keyref` under the root node.

```xml
<xsd:key name="Ifc2x_final.Organization">
  <xsd:selector xpath="Organization"/>
  <xsd:field xpath="@id"/>
</xsd:key>

<xsd:keyref name="Ifc2x_final.Organization-ref" refers="Ifc2x_final.Organization">
  <xsd:selector xpath=".//Organization"/>
  <xsd:field xpath="@href"/>
</xsd:keyref>
```

**NOTE** The `xsd:key` creates a node set within the uppermost containing element, the document node. This element is the root element of the XML document. The EXPRESS schema declaration translates into the XSD root element declaration and contains all `xsd:key` and `xsd:keyref` pairs of the document.

### 3.6.5 Mapping of explicit attributes

An explicit attribute represents a property of the entity in which scope it is declared:

- it shall have a value inserted for each valid entity instance, or
- it may have a value in case of an optional explicit attribute

The base type of an attribute could be a simple data type, a non-constructed defined data type, a constructed data type or an entity data type. Within the EXPRESS definitions, the order of the attribute declarations are not significant, however after mapping the EXPRESS structure into the exchange structure of the ISO10303-21 SPF, the order of attributes become significant.

**NOTE** This is required as the attribute identifiers are not contained within an SPF exchange file and the mapping of the attribute values to its attribute identifiers is done based on the sequence of the attributes within the exchange file.

In the XML instance document, attribute content is not ordered, whereas element content might be ordered or unordered according to the declaration within the `complexType`.

The `sequence` constraint imposes an order among the parts within an element content, whereas the `all` constraint does not imply an order. However the implication of no order for `all` only applies to the `complexType` within it is declared, not to the `complexContent` over the whole type hierarchy. In addition, the `all` constraint imposes the rule, that no `element` within the element content may appear more than once.

Compared with the EXPRESS attribute definition rules, the SPF exchange rules and the restrictions of the `all` declaration, the element content should be declared with the `sequence` constraint.

#### 3.6.5.1 Mapping of explicit attributes with simple or defined base type

For each EXPRESS explicit attribute (either mandatory or optional) which has a base data type being either a simple data type or a non-constructed defined data type or a constructed defined data type of ENUMERATION the corresponding `complexType` (see clause 3.6) should:

- Contain an element content with `sequence` constraint of `element` descriptions, with the `name` attribute having as its value the EXPRESS attribute identifier. The name shall begin with a lower capital letter. The `sequence` shall imply an order, being the order of attributes within the ENTITY definition.
If the EXPRESS explicit attribute is mandatory, the element description shall have the `minOccurs` particle with the value "1". Since the default value for `minOccurs` is 1, the particle may be omitted.

If the EXPRESS attribute is optional, the element description shall have the `minOccurs` particle with the value "0".

Considering the following EXPRESS definition:

``` EXPRESS
ENTITY IfcOrganization;
  Name : IfcLabel;
  Description : OPTIONAL IfcText;
END_ENTITY;
```

This should translate into the following XML schema definition (only the `complexType` is shown):

```xml
<xsd:complexType name="Organization">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element type="Label" name="name"/>
        <xsd:element type="Text" name="description" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

And to a sample XML document:

```xml
<Organization id="_100">
  <name>International Alliance for Interoperability</name>
  <description>Organization defining the IFC model.</description>
</Organization>
```

Or to a sample XML document, like:

```xml
<Organization id="_100">
  <name>International Alliance for Interoperability</name>
</Organization>
```

### 3.6.5.2 Mapping of explicit attributes with select base type

For each EXPRESS explicit attribute (either mandatory or optional) which has a base data type being a SELECT constructed datatype the corresponding `complexType` (see clause 3.6) shall:

- include an element content with `sequence` constraint of local element declarations with the `name` attribute having as its value the EXPRESS attribute identifier (starting with lower case letter),
- each local element shall have an underlying `complexType` declaration, referring to the global group, created for all SELECT types (see clause 3.5.2).

Considering the following EXPRESS definition:

``` EXPRESS
ENTITY IfcActor
  TheActor : IfcActorSelect;
END_ENTITY;
```

This should translate into the following XML schema definition (only the `complexType` is shown):

```xml
<xsd:complexType name="Actor">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
A sample XML document part could be (showing single and multi-references):

```xml
<Actor id="_1003">
  <theActor>
    <Person>
      <givenName>Thomas</givenName>
      <familyName>Liebich</familyName>
    </Person>
  </theActor>
</Actor>

<Actor id="_1004">
  <theActor>
    <Organization xsi:nil="true" href="_1002"/>
  </theActor>
</Actor>

<Organization id="_1002">
  <name>International Alliance for Interoperability</name>
</Organization>
```

3.6.5.3 Mapping of explicit attributes with entity base type

EXPRESS resolves all relationships to other entities as entity references. There is no distinction between containment by value or by reference. XML has both concepts, containment by value using the containment structure and containment by reference, using the ID/IDREF, or key/keyref, or both definitions.

The EXPRESS to XML schema language binding should describe both possibilities (containment and reference), following the SOAP approach, whereas its implementation may have to choose one or has to introduce further restrictions, such as using the EXPRESS comments to inform the translator.

**NOTE** Depending on the context of the EXPRESS definition, either only reference or only containment could not be appropriate for the translation, as the decision needs to be done on an ENTITY by ENTITY basis. This requires an additional optimization step. This case-by-case decision, however, is currently out of scope of the ifcXML language binding.

For each EXPRESS explicit attribute (either mandatory or optional) which has a base data type being an entity data type the corresponding `complexType` (see clause 3.6) shall:

- Contain an element content with `sequence` constraint of `element` descriptions, with the `name` attribute having as its value the EXPRESS attribute identifier. The name shall begin with a lower capital letter,
- Each of the `element` declarations shall define a `complexType` including the `element` reference to the global `element` defining the `ENTITY` data type.

In order to provide for the empty embedded element as accessor of multi-reference independent elements, the local element declaration shall be `nillable`. By default each element declaration does contain the attribute `nillable="true"`. This allows the instantiation as accessor element (or referencing element) without any content given.
For "containmentType" being "containmentOnly", it should translate into XML schema definition with nillable="false", since empty content elements would be prohibited by the demand for containment. Since "false" is the default value for nillable the attribute can be omitted.

Considering the following EXPRESS definition (from ISO 10303-41 person_and_organization):

```plaintext
ENTITY IfcPersonAndOrganization;
  ThePerson : IfcPerson;
  TheOrganization : IfcOrganization;
END_ENTITY;
```

This should translate into the following XML schema definition (only the complexType is shown), here in the case of "containmentType" being "choice" or "referenceOnly".

```xml
<xsd:complexType name="PersonAndOrganization">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="thePerson">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element ref="Person"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="theOrganization">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element ref="Organization"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The following XML instance document includes an example using single reference

```xml
<PersonAndOrganization id="_1000">
  <thePerson>
    <Person>
      <name>Grobler</name>
    </Person>
  </thePerson>
  <theOrganization>
    <Organization>
      <name>IAI</name>
    </Organization>
  </theOrganization>
</PersonAndOrganization>
```

The next XML instance document includes an example using multi-reference

```xml
<PersonAndOrganization id="_1000">
  <thePerson>
    <Person href="_1001"/>
  </thePerson>
  <theOrganization>
    <Organization href="_1002"/>
  </theOrganization>
</PersonAndOrganization>
```
The next XML instance document includes an example mixing the use of single and multi-reference

```xml
<Person id="_1001">
  <name>Grobler</name>
</Person>

<Organization id="_1002">
  <name>IAI</name>
</Organization>
```

### 3.6.6 Mapping of explicit attributes that are aggregates

For each EXPRESS explicit attribute (either mandatory or optional) which has a base data type being an aggregate of the underlying type, the corresponding complexType (see clause 3.6) shall be defined according to the underlying data type. The aggregation shall be translated by using the particles minOccurs and maxOccurs.

#### 3.6.6.1 Mapping of explicit attributes with aggregated simple base type

If the underlying type is a value based simple type (number, real, integer, binary, Boolean, or Logical), or a non-constructed defined datatype, based on a value based simple type, or an ENUMERATION datatype, than the XML type shall be a local restriction of the list data type for the value based simpleType given.

Considering the following EXPRESS definition (quoted from ISO 10303-41 cartesian_point):

```plaintext
ENTITY IfcCartesianPoint
  Coordinates : LIST [1:3] OF IfcLengthMeasure;
END_ENTITY;
```

The XML schema definition shall be the following:

```xml
<xsd:complexType name="CartesianPoint">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="coordinates">
          <xsd:simpleType>
            <xsd:restriction>
              <xsd:list itemType="LengthMeasure"/>
            </xsd:simpleType>
          </xsd:restriction>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
The XML instance document looks like:

```xml
<CartesianPoint id="1006">
  <coordinates>1.5 1.0 0.0</coordinates>
</CartesianPoint>
```

The mapping is done by locally declaring a `simpleType`, which references the `simpleType` with the `list` declaration of the base type and restricts list bounds by `minOccurs` and `maxOccurs` particles.

### 3.6.6.2 Mapping of explicit attributes with aggregated string type

If the explicit EXPRESS attribute is an aggregate of string, or a non-constructed defined data type based on string, it should result into a local `element` declaration, which encloses an aggregate of string based elements.

- The local `element` shall have the `name` attribute with its value being the EXPRESS attribute identifier and shall have the `minOccurs="0"` attribute, if the EXPRESS attribute is declared to be `OPTIONAL`.
- The `element` includes a `complexType` definition with constraint `choice`, which declares a synthetic `element` with the `name="String"` of `type=xsd:string` (or the type of the `simpleType` being a restriction of `xsd:string`).
- the constraint `choice` shall have the `minOccurs` attribute with the value of the lower bound of the EXPRESS aggregate (maybe omitted, if the value is "1") and the `maxOccurs` attribute with the value of the upper bound of the EXPRESS aggregate. If the upper bound of the EXPRESS aggregate is undetermined (i.e. "?"), then the value of the `maxOccurs` shall be "unbounded".

**NOTE** The string based explicit attributes can not be translated into a restriction of the `xsd:list` type, since strings holding white spaces within the string value would be interpreted as lists.

Considering the following EXPRESS definition (quoted from ISO 10303-41 person):

```xml
ENTITY IfcPerson;
  GivenName : IfcLabel;
  FamilyName : IfcLabel;
  MiddleNames : OPTIONAL LIST [1:?] OF IfcLabel;
END_ENTITY;
```

**Using the translation into XML elements within the `complexType` it results**

```xml
<xsd:complexType name="Person">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element type="Label" name="givenName"/>
        <xsd:element type="Label" name="familyName"/>
        <xsd:element minOccurs="0" name="middleNames">
          <xsd:complexType>
            <xsd:choice maxOccurs="unbounded" minOccurs="1">
              <xsd:element type="Label" name="String"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
In an XML instance document it looks like

```xml
<Person id="_100">
  <familyName>Maier</familyName>
  <givenName>Hans</givenName>
  <middleNames>
    <String>Dieter</String>
    <String>Wilhelm</String>
    <String>Friedrich</String>
  </middleNames>
</Person>
```

3.6.6.3 Mapping of explicit attributes with aggregated entity data type

If the explicit EXPRESS attribute is an aggregate of an entity or select data type, the resulting complexType definition of the element description shall be an aggregate.

- The local element shall have the name attribute with its value being the EXPRESS attribute identifier (or role name) and shall have the minOccurs="0" attribute, if the EXPRESS attribute is declared to be OPTIONAL.
- The element includes a complexType definition with constraint choice, which declares a local element referencing the global element declaration for the aggregated attribute with entity datatype (see 0).
- the constraint choice shall have the minOccurs attribute with the value of the lower bound of the EXPRESS aggregate (maybe omitted, if the value is "1") and the maxOccurs attribute with the value of the upper bound of the EXPRESS aggregate. If the upper bound of the EXPRESS aggregate is undetermined (i.e. "?"), than the value of the maxOccurs shall be "unbounded".

Considering the following EXPRESS definition:

```plaintext
ENTITY IfcPerson;
  -- additional attributes skipped
  Addresses : OPTIONAL LIST [1:?] OF IfcAddress;
END_ENTITY;
```

The corresponding XML schema definition is (in case of default containmentType):

```xml
<xsd:complexType name="Person">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element minOccurs="0" name="addresses">
          <xsd:complexType>
            <xsd:choice maxOccurs="unbounded" minOccurs="1">
              <xsd:element ref="Address"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

And the XML instance document:

```xml
<Person id="_100">
  <addresses>
    <PostalAddress href="_101"/>
    <TelecomAddress href="_102"/>
  </addresses>
</Person>
```
The \textit{minOccurs="0"} at the element definition of the role \textit{addresses} corresponds to the \textbf{OPTIONAL} declaration of the \textit{EXPress} attribute \textit{Addresses}, whereas the \textit{minOccurs="1"} (maybe omitted since default) and \textit{maxOccurs="unbounded"} at the \textit{choice} statement corresponds to the \textit{EXPress} \textit{LIST} bounds.

3.6.7 Mapping of Inverse attributes

If another entity has established a relationship with the current entity by the way of an explicit attribute, an inverse attribute may be used to describe that relationship in the context of the current entity. (See also ISO 10303-11 clause 9.2.1.3).

The \textbf{INVERSE} relationship always exists (even without declaring it as an inverse attribute), and the inverse relationship are not explicitly exchanged in an ISO 10303-21 SPF.

\textit{INVERSE} relationships are always handled be reference, and for convenience of the receiver, it should be possible to exchange them in an XML instance document. However, if the current element (translated from the entity that has the inverse relationship) is used in the containment structure of the other element, the use of an inverse relationship is obsolete.

\textit{INVERSE} attributes are aggregates (of type \textit{SET} or \textit{BAG}) by definition. Therefore they could be translated into XML attribute declarations with the data type \textit{xsd:IDREFS}. This translation would lead to a "lean" equivalent in XML schema and XML instance documents. However, it would lack the consistency checks otherwise provided by the \textit{key} and \textit{keyref} structure and the direct information of the referenced data type in the XML instance document. In particular it introduces a danger of inconsistency.

To balance the advantages and disadvantages, the translation of \textit{INVERSE} attributes into the corresponding XML schema declarations should be guided by configuration. The configuration tag \textit{"inverseAttributes"} (see 3.2.2) can be set to:

- \textit{"exclude"} (the default setting) – then no \textit{INVERSE} attributes are explicitly translated into the XML schema definition. All inverse information has to be reconstructed, after receiving the XML instance document.
- \textit{"include"} – then all \textit{INVERSE} attributes are translated into local XML \textit{element} declarations within the \textit{complexType} corresponding to the \textit{ENTITY} definition.

\textbf{NOTE} If the configuration tag \textit{"containmentType"} is set to \textit{"containmentOnly"}, no \textit{INVERSE} attributes shall be translated independent of the setting of the configuration tag \textit{"inverseAttributes"}.

3.6.7.1 Detailed mapping of inverse attributes

Given the setting of the configuration tag \textit{"inverseAttributes"} to \textit{"include"}, the following translations shall occur to all \textit{INVERSE} attributes in the \textit{EXPress} definition.

- a global \textit{attribute} is declared with \textit{name="inverse"} and \textit{type="xsd:IDREFS"}, this global \textit{attribute} is later assigned to all element declarations for \textit{INVERSE} attributes
- a local \textit{element} is declared within the \textit{complexContent} of the \textit{complexType} referring to the element declaration,
  - it shall have the \textit{name} attribute with its value being the \textit{INVERSE} attribute identifier (beginning with lower capital letter),
  - it shall have a \textit{complexType} definition, which assigns the global \textit{attribute} \textit{"inverse"} by using \textit{ref},
  - if the lower bound of the \textit{INVERSE} attribute is \textit{"0"}, then the local \textit{element} declaration shall include the attribute \textit{minOccurs="0"}, otherwise no \textit{minOccurs} attribute is given.

\textit{Considering the following EXPress definition:}
ENTITY IfcRelAssignsToActor
    SUBTYPE OF (IfcRelAssigns);
    RelatingActor : IfcActor;
END_ENTITY;

ENTITY IfcActor
    SUBTYPE OF (IfcObject);
    TheActor : IfcActorSelect;
    INVERSE
        IsActingUpon : SET OF IfcRelAssignsToActor FOR RelatingActor;
END_ENTITY;

It would lead to the following XML schema definition, based on the default configuration tags "containmentType" and "inverseAttributes" set to "include":

```xml
<xsd:attribute name="inverse" type="xsd:IDREFS"/>
<xsd:complexType name="RelAssignsToActor">
    <xsd:complexContent>
        <xsd:extension base="RelAssigns">
            <xsd:sequence>
                <xsd:element name="relatingActor">
                    <xsd:complexType>
                        <xsd:choice>
                            <xsd:element ref="Actor"/>
                        </xsd:choice>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="isActingUpon" minOccurs="0">
                    <xsd:complexType>
                        <xsd:attribute ref="inverse"/>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Actor">
    <xsd:complexContent>
        <xsd:extension base="Object">
            <xsd:sequence>
                <xsd:element name="theActor">
                    <xsd:complexType>
                        <xsd:choice>
                            <xsd:group ref="ActorSelect"/>
                        </xsd:choice>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="isActingUpon" minOccurs="0" minOccurs="0">
                    <xsd:complexType>
                        <xsd:attribute ref="inverse"/>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

And to an XML instance like:

```xml
<Actor id="_100">
    <theActor>
        <Organization href="_102"/>
    </theActor>
    <isActingUpon inverse="_101"/>
</Actor>
```
3.6.8 Mapping of derived attributes

**DERIVE** attributes are used within EXPRESS definitions to derive the value for a property by computing it from other attribute values. The expression for the computation is given for the **DERIVE** attributes together with the datatype.

Once declared, **DERIVE** attributes always have to have a value (can not be optional), but the value is recalculated after the exchange and not explicitly included in the exchange file. Therefore **DERIVE** attributes are not mapped into the exchange structure of an ISO 10303-21 SPF.

The XML schema definition does not allow to capture the expressions, that would allow the calculation of **DERIVE** attributes from other attribute values. There are two scenarios to handle **DERIVE** attributes:

- all **DERIVE** attributes are excluded from being mapped into the XML schema definitions and thereby from being exchanged in an XML instance document. The advantage is, that redundant and possibly inconsistent information is exchanged. The disadvantage is, that the receiver would need to recalculate the values with being informed about the expressions by the XML specification.
- all **DERIVE** attributes are included in the mapping, and the XML instance document allows the exchange of the (pre-)calculated values of the **DERIVE** attributes. The advantage is that all values are there, however the exchange structure is redundant and maybe inconsistent.

To balance the advantages and disadvantages, the translation of **DERIVE** attributes into the corresponding XML schema declarations should be guided by configuration. The configuration tag "derivedAttributes" (see 3.2.2) can be set to:

- "exclude" (the default setting) – then no **DERIVE** attributes are explicitly translated into the XML schema definition. All derived information has to be calculated, after receiving the XML instance document.
- "include" – then all **DERIVE** attributes are translated into local XML element declarations within the complexType corresponding to the **DERIVE** attribute within the ENTITY definition.

### 3.6.8.1 Detailed mapping of derived attributes

Given the setting of the configuration tag "derivedAttributes" to "include", the following translations shall occur to all INVERSE attributes in the EXPRESS definition.

- a local element is declared within the complexContent of the complexType referring to the element declaration,
  - it shall have the name attribute with its value being the **DERIVE** attribute identifier,
  - it shall have the type attribute being the type of the **DERIVE** attribute, the mapping of the attribute type shall follow the mapping rules for mandatory non-aggregated attributes (see 3.6.5), or for aggregated attributes (see 3.6.6).

### 3.6.9 Mapping of Sub / Supertypes

EXPRESS allows for a very complex inheritance structure, including multiple inheritance and non-exclusive subtypes. XML schema allows for the derivation of new types from existing types (by using either extension or restriction). Then a complexType has to be declared with complexContent. Since the XML schema technical recommendation also allows a substitution of elements by other elements in an instance document using the substitutionGroup, both concepts (the complexContent and substitutionGroup) can be used to represent simple inheritance the (where the inheritance graph is restricted to be a tree) within the XML schema definitions.

The inheritance structure within the IFC model is restricted to single, exclusive inheritance. Therefore the inheritance graph in IFC is always a tree. This allows to fully translate the inheritance structure of the IFC model into XML schema definitions.
NOTE  The method described in the ifcXML language binding is limited to single inheritance without any complex entity types. Solutions beyond this limitation are out of scope of this version of the language binding document.

3.6.9.1 Mapping of single inheritance structures without redeclarations

For all entity data subtypes that do not redeclare attribute definitions of its supertypes, the translation shall be provided as:

- substitutionGroup for the element definition
- complexContent for the complexType definitions for the element definitions, and the base attribute of the complexContext shall be the type name of the supertype and should include the extension declaration.

This leads to a type and substitution hierarchy, which reflect the subtype hierarchy in EXPRESS (restricted to single inheritance).

Considering the following EXPRESS definition:

```express
ENTITY IfcAddress
  ABSTRACT SUPERTYPE OF (ONEOF(IfcPostalAddress, IfcTelecomAddress));
  Purpose : OPTIONAL IfcAddressTypeEnum;
  Description : OPTIONAL IfcText;
END_ENTITY;

ENTITY IfcPostalAddress
  SUBTYPE OF (IfcAddress);
  Town : OPTIONAL IfcLabel;
  PostalCode : OPTIONAL IfcLabel;
  -- other attributes skipped
END_ENTITY;
```

The IfcAddress does not have a supertype. It therefore derives its type directly from element. As it is an abstract supertype, the abstract attribute is inserted.

The XML schema definition should be defined for the supertype IfcAddress as:

```xml
<xsd:element name="Address" type="Address" abstract="true"
nillable="true"/>
</xsd:element>

<xsd:complexType name="Address" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="purpose" type="AddressEnum"
          minOccurs="0"/>
        <xsd:element name="description" type="Label"
          minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

For the subtype IfcPostalAddress it would be:

```xml
<xsd:element name="PostalAddress" type="PostalAddress"
  substitutionGroup="Address" nillable="true"/>
</xsd:element>

<xsd:complexType name="PostalAddress">
  <xsd:complexContent>
    <xsd:extension base="Address">
      <xsd:sequence>
        ...
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
The mapping of the EXPRESS inheritance structure create an XML type hierarchy and an XML substitution structure. Both are required to be a tree graph. The next Figure 3 shows the hierarchy of the Address element declaration.

![Figure 3: Type and substitution hierarchy for element declarations](image)

The type hierarchy satisfies the XML substitutionGroup requirement, that the element, that substitutes another element, needs to have the same type, or a derived type. A xsd:key xsd:keyref constraint definition schema components however has to be included within the root element declaration (see also 3.2.1). Extending the example above:

```xml
<xsd:element name="ifcXML">
  <!other definitions -->
  <xsd:element name="IIfc2X_platform.PostalAddress">
    <xsd:selector xpath="IIfc2X_platform.PostalAddress"/>
    <xsd:field xpath="@id"/>
  </xsd:element>
</xsd:element>
```

### 3.6.9.2 Mapping of single inheritance structures with redeclarations

Attributes declared in a supertype can be redeclared within its subtypes according to the EXPRESS language specification. The redeclaration can be used to further constrain the datatype of the attribute, or its cardinality, to change optional into mandatory use or explicit into derived use.

**NOTE** XML schema allows the use of restriction within an complexContent to restrict the derived base type. However in contrary to EXPRESS, where redeclarations and new
attributes can be defined within the same subtype, the complexContent within a complexType declaration can only be either by restriction or by extension. Since restriction also allows the addition of new element’s, restriction should be used for subtypes which contain redeclarations.

IAI-NOTE The IFC modeling guidelines discourage the use of redeclarations of attribute types in order to comply to the "Liskov principle". However within those parts, that had been adapted from ISO 10303, like the unit definitions, redeclarations appear in the IFC2x model.

For all entity data subtypes that do redeclare attribute definitions of its supertypes, the translation shall be provided as:

- substitutionGroup for the element definition
- complexContent for the complexType definitions for the element definitions, and the base attribute of the complexContext shall be the type name of the supertype and should include the restriction declaration.

Within the restriction declaration all inherited attributes (mapped into element's) shall be repeated, for all non redeclared attributes, the mapping should be repeated unchanged. For redeclared explicit attributes, the following shall be done

- if the redeclared attribute is an explicit attribute, the mapping should follow the instructions imposed by the redeclared datatype,
- if the redeclared attribute is a derived attribute, the mapping depends on the setting of the configuration tag "derivedAttributes" (see 3.2.2):
  - "exclude" (the default setting) – then no element declaration shall be generated for the DERIVE attribute. It is therefore omitted from the subtype (similar to the "*" mark within the STEP physical file exchange structure.
  - "include" – then an element declaration shall be generated for the DERIVE attribute. It is translated into local XML element declaration within the complexType corresponding to the rule applying to the datatype of the redeclared DERIVE attribute.

Considering the following EXPRESS definition:

```express
ENTITY IfcNamedUnit
  ABSTRACT SUPERTYPE OF (ONEOF(IfcSIUnit));
  Dimensions : IfcDimensionalExponents;
  UnitType : IfcUnitEnum;
END_ENTITY;

ENTITY IfcSIUnit
  SUBTYPE OF (IfcNamedUnit);
  Prefix : OPTIONAL IfcSIPrefix;
  Name : IfcSIUnitName;
  DERIVE
    SELF\IfcNamedUnit.Dimensions
    : IfcDimensionalExponents := IfcDimensionsForSiUnit (SELF.Name);
END_ENTITY;
```

It should result to the following XML schema definition, based on the setting of the configuration tag "derivedAttributes" to "exclude".

```xml
<xsd:complexType name="NamedUnit" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="dimensions">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element ref="DimensionalExponents"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
An XML document could now contain the following instance of IfcSIUnit (omitting the redeclared derived attribute DimensionalExponents):

```xml
<SIUnit>
  <unitType>lengthunit</unitType>
  <prefix>milli</prefix>
  <name>metre</name>
</SIUnit>
```
4 Example of ifcXML

The following example shows the mapping of the unit and measure resource of IFC2x, the IfcMeasureResource, into XML schema. It shows the original IFC EXPRESS definitions, its mapping into XML schema, and sample files, both in STEP physical file and XML.

IAI-NOTE A slightly reduced version of the original IFC2x IfcMeasureResource had been used in that example for the sake of simplicity for the demonstrated sample files.

4.1 Unit Definition and Assignment

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

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

4.1.1 Introduction

In different types of units there are four basic cases:

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

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

- The data type of an entity attribute is a measure defined type as such. In this case it’s the global unit assignment for the corresponding unit for this measure type that defines the unit for all the usages of this defined measure type. The definition of the units is handled through the IfcUnitAssignment, that has to be included in the exchange file (normally referenced from IfcProject).
- The data type of an attribute is IfcMeasureWithUnit, which allows for definition of unit per instance of that entity type, independent of global unit assignment;

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

IAI-NOTE The different defined measure types are not shown in the EXPRESS and XSD examples, for simplicity reasons of easy demonstration of XSD syntax, the IfcValue SELECT had been restricted to IfcInteger, IfcReal, IfcLabel.
4.1.2 EXPRESS definition

The global unit assignment of the project defines the global units for measures and values when the units are not otherwise defined more specifically using entity type *IfcMeasureWithUnit* as attribute's data type. The relevant entity within the *IfcMeasureResource*, which is used to list all globally defined units, is the *IfcUnitAssignment*.

![Diagram of IfcMeasureResource]

*Figure 4: Definition of the IfcMeasureResource*

The Figure 4 above shows the EXPRESS_G diagram, depicting the definitions for *IfcMeasureResource*. The listing below shows the EXPRESS code of the *IfcMeasureResource* (the EXPRESS FUNCTION's and WHERE rules are omitted for simplicity reasons, as they do not translate into XML schema).
SCHEMA IFCMESUREMENTRESOURCE;

TYPE IfcInteger = INTEGER;
END_TYPE;

TYPE IfcLabel = STRING;
END_TYPE;

TYPE IfcReal = REAL;
END_TYPE;

TYPE IfcDerivedUnitEnum = ENUMERATION OF
  (ANGULARVELOCITYUNIT,
   COMPOUNDPLANEANGLEUNIT,
   DYNAMICVISCOITYUNIT,
   HEATFLUXDENSITYUNIT,
   INTERCOURTIREUNIT,
   ISOHERMALMOISTURECAPACITYUNIT,
   KINETICVISCOITYUNIT,
   LINEARVELOCITYUNIT,
   MASSSENSIVITYUNIT,
   MASSFLOWRATEUNIT,
   MOISTUREDIFFUSIVITYUNIT,
   MOLECULARWEIGHTUNIT,
   SPECIFICHEMICALCAPACITYUNIT,
   THERMALACTIVETEMPUNIT,
   THERMALCONDUCTANCEUNIT,
   THERMALRESISTANCEUNIT,
   THERMALTRANSMITTANCEUNIT,
   VAPORPERMEABILITYUNIT,
   VOLUMETRICFLOWRATEUNIT,
   ROTATIONALFREQUENCYUNIT,
   TORQUEUNIT,
   MOMENTORINERTIUNIT,
   LINEARMOMENTUNIT,
   LINEARFORCEUNIT,
   PLANARFORCEUNIT,
   MODULUSOFELASTICITYUNIT,
   SHEARMODULUSUNIT,
   LINEARSTIFFNESSUNIT,
   MODULUSOFSUBGRADEREACTIONUNIT,
   ACCELERATIONUNIT,
   USERDEFINED);
END_TYPE;

TYPE IfcPrefix = ENUMERATION OF
  (EXA , PETA , TERA , GIGA , MEGA , KILO,
   HECTO , DECA , DECI , CENTI , MILLI,
   MICRO , NANO , PICO , FEMTO , ATTO);
END_TYPE;

TYPE IfcUnitName = ENUMERATION OF
  (AMPERE , BICYCLE , CANDELA,
   CURB , CUBIC_METER , DEGREE_CELSIUS,
   FARAD , GRAM , GRAY,
   HENRY , HERTZ , JOULE,
   KELVIN , LUMEN , LUX,
   METER , MOLE , NEWTON,
   OHM , PASCAL , RADIAN,
   SECOND , SIEMENS , SIERT,
   SQUARE_METER , STERADIAN , TESLA,
   VOLT , WATT , WEBER);
END_TYPE;

TYPE IfcUnitsEnum = ENUMERATION OF
  (ABSORBEDDOSEUNIT,
   AMOUNTOFSUBSTANCEUNIT,
   AREAUNIT,
   DOSEEQUIVALENTUNIT,
   ELECTRICCAPACITANCEUNIT,
   ELECTRICCURRENTUNIT,
   ELECTRICCONDUCTANCEUNIT,
   ELECTRICRESISTANCEUNIT,
   ELECTRICVOLTAGEUNIT,
   ENERGYUNIT,
   FORCEUNIT,
   FREQUENCYUNIT,
   ILLUMINANCEUNIT,
   INDUCTANCEUNIT,
   LENGTHUNIT,
   LIGHTUNIT,
   LUMINOUSFLUXUNIT,
   LUMINOUSINTENSITYUNIT,
   MAGNETICEFLUXDENSITYUNIT,
   MAGNETICEFLUXUNIT,
   MASSUNIT,
   PLANEMEASUREUNIT,
   POWERUNIT,
   PRESSUREUNIT,
   RADIANTUNIT,
   RADIOACTIVITYUNIT,
   RADIATIONUNIT,
   THERMALDYNAMICTEMPERATUREUNIT,
   TIMEUNIT,
   VOLUMEUNIT,
   USERDEFINED);
END_TYPE;

TYPE IfcUnit = SELECT
  (IfcDerivedUnit,
   IfcNamedUnit);
END_TYPE;

TYPE IfcValue = SELECT
  (IfcInteger,
   IfcReal,
   IfcLabel);
END_TYPE;

ENTITY IfcContextDependentUnit
  SUBTYPE OF (IfcNamedUnit);
  Name : IfcLabel;
END_ENTITY;

ENTITY IfcConversionBasedUnit
  SUBTYPE OF (IfcNamedUnit);
  Name : IfcLabel;
  ConversionFactor : IfcMeasureWithUnit;
END_ENTITY;

ENTITY IfcDerivedUnit;
  Elements : SET [1:?] OF
    IfcDerivedUnitElement;
  UnitType : IfcDerivedUnitsEnum;
  UserDefinedType : OPTIONAL IfcLabel;
  DERIVE
    Dimensions : IfcDimensionalExponents :=
      IfcDerivedDimensionalExponents(SELF);
END_ENTITY;

ENTITY IfcDerivedUnitElement;
  Unit : IfcNamedUnit;
  Exponent : INTEGER;
END_ENTITY;

Editor: Thomas Liebich  11.12.2001  60 of 76
ENTITY IfcDimensionalExponents;
    LengthExponent : INTEGER;
    MassExponent : INTEGER;
    TimeExponent : INTEGER;
    ElectricCurrentExponent : INTEGER;
    ThermodynamicTemperatureExponent : INTEGER;
    AmountOfSubstanceExponent : INTEGER;
    LuminousIntensityExponent : INTEGER;
END_ENTITY;

ENTITY IfcMeasureWithUnit;
    ValueComponent : IfcReal;
    UnitComponent : IfcUnit;
END_ENTITY;

ENTITY IfcNamedUnit
ABSTRACT SUPERTYPE OF (ONEOF
    IfcContextDependentUnit,
    IfcConversionBasedUnit,
    IfcSIUnit);
    Dimensions : IfcDimensionalExponents;
    UnitType : IfcUnitEnum;
    WHERE
        WR1 : IfcCorrectDimensions (SELF.UnitType,
            SELF.Dimensions);
END_ENTITY;

ENTITY IfcSIUnit
SUBTYPE OF (IfcNamedUnit);
    Prefix : OPTIONAL IfcSIPrefix;
    Name : IfcSIUnitName;
    DERIVE
        SELF\IfcNamedUnit.Dimensions :
            IfcDimensionalExponents :=
                IfcDimensionsForSIUnit (SELF.Name);
END_ENTITY;

ENTITY IfcUnitAssignment;
    Units : SET [1:?] OF IfcUnit;
END_ENTITY;

END_SCHEMA;

4.1.3 XML schema translation

After translation into XML schema, all non-abstract ENTITY’s are translated as independent elements within the root element (name taken from the original EXPRESS schema – here the IfcMeasureResource).

![Figure 5: XSD declarations of independent elements](image)

Than the independent ELEMENT’s are further declared by element content, following the EXPRESS attribute definitions. A graphical representation is shown in below.
Figure 6: XSD Definition of IfcMeasureResource
The XSD listing shows first the declaration of name spaces and the root element, also containing the `xsd:key` and `xsd:keyref` constraints. Configuration tags are omitted, since the default settings apply.

```xml
<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns="http://www.iai-international.org/ifcXML/ifc2x/IfcMeasureResource"
xmlns:ifc2x="http://www.iai-international.org/ifcXML/ifc2x/IfcMeasureResource"
targetNamespace="http://www.iai-international.org/ifcXML/ifc2x/IfcMeasureResource"
xmlns:xlink="http://www.w3.org/2001/xlink"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- root element declaration (for SCHEMA definitions) -->
  <xsd:element name="ifcXML">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element ref="ContextDependentUnit"/>
        <xsd:element ref="ConversionBasedUnit"/>
        <xsd:element ref="DerivedUnit"/>
        <xsd:element ref="DerivedUnitElement"/>
        <xsd:element ref="DimensionalExponents"/>
        <xsd:element ref="MeasureWithUnit"/>
        <xsd:element ref="SIUnit"/>
        <xsd:element ref="UnitAssignment"/>
      </xsd:choice>
    </xsd:complexType>
    <xsd:attributeGroup ref="configTypes"/>
    <xsd:attributeGroup ref="configTags"/>
  </xsd:complexType>
  <xsd:key name="IfcMeasureResource.ContextDependentUnit">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key ref="IfcMeasureResource.ContextDependentUnit" name="IfcMeasureResource.ContextDependentUnit-ref">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key name="IfcMeasureResource.ConversionBasedUnit">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key ref="IfcMeasureResource.ConversionBasedUnit" name="IfcMeasureResource.ConversionBasedUnit-ref">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key name="IfcMeasureResource.DerivedUnit">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key ref="IfcMeasureResource.DerivedUnit" name="IfcMeasureResource.DerivedUnit-ref">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key name="IfcMeasureResource.DerivedUnitElement">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key ref="IfcMeasureResource.DerivedUnitElement" name="IfcMeasureResource.DerivedUnitElement-ref">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key name="IfcMeasureResource.DimensionalExponents">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key ref="IfcMeasureResource.DimensionalExponents" name="IfcMeasureResource.DimensionalExponents-ref">
    <xsd:selector xpath="@id"/>
  </xsd:key>
  <xsd:key name="IfcMeasureResource.MeasureWithUnit">
    <xsd:selector xpath="@id"/>
  </xsd:key>
</xsd:element>
</xsd:schema>
```
Then all element declarations are given for the EXPRESS ENTITY, including the declarations for abstract element's, that do not appear under the root element.

```
<!-- element declarations (for ENTITY definitions) -->
<xsd:element name="ContextDependentUnit" type="ContextDependentUnit"
substitutionGroup="NamedUnit" nillable="true"/>
<xsd:element name="ConversionBasedUnit" type="ConversionBasedUnit"
substitutionGroup="NamedUnit" nillable="true"/>
<xsd:element name="DerivedUnit" type="DerivedUnit" nillable="true"/>
<xsd:element name="DerivedUnitElement" type="DerivedUnitElement" nillable="true"/>
<xsd:element name="DimensionalExponents" type="DimensionalExponents" nillable="true"/>
<xsd:element name="NamedUnit" type="NamedUnit" abstract="true" nillable="true"/>
<xsd:element name="SIUnit" type="SIUnit" substitutionGroup="NamedUnit" nillable="true"/>
<xsd:element name="UnitAssignment" type="UnitAssignment" nillable="true"/>
```

Now the complexType's are declared for all element declarations for the EXPRESS ENTITY. It contains the EXPRESS attribute definitions and allows for empty elements (xsi:nil="true") and for embedded elements. Therefore both containment and reference is supported. All inherit from the standard base element, which declares the id and href attributes.

```
<!-- complex types for element declarations (for ENTITY definitions) -->
<xsd:complexType name="ContextDependentUnit">
  <xsd:complexContent>
    <xsd:extension base="NamedUnit">
      <xsd:sequence>
        <xsd:element name="name" type="Label"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ConversionBasedUnit">
  <xsd:complexContent>
    <xsd:extension base="NamedUnit">
      <xsd:sequence>
        <xsd:element name="name" type="Label"/>
        <xsd:element name="conversionFactor">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element ref="MeasureWithUnit"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
<xsd:complexType name="DerivedUnit">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="elements">
          <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
              <xsd:element ref="DerivedUnitElement"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="unitType" type="DerivedUnitEnum"/>
        <xsd:element name="userDefinedType" type="Label" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="DerivedUnitElement">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="unit">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element ref="NamedUnit"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="exponent" type="xsd:long"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="DimensionalExponents">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="lengthExponent" type="xsd:long"/>
        <xsd:element name="massExponent" type="xsd:long"/>
        <xsd:element name="timeExponent" type="xsd:long"/>
        <xsd:element name="electricCurrentExponent" type="xsd:long"/>
        <xsd:element name="thermodynamicTemperatureExponent" type="xsd:long"/>
        <xsd:element name="amountOfSubstanceExponent" type="xsd:long"/>
        <xsd:element name="luminousIntensityExponent" type="xsd:long"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="MeasureWithUnit">
  <xsd:complexContent>
    <xsd:extension base="element">
      <xsd:sequence>
        <xsd:element name="valueComponent">
          <xsd:complexType>
            <xsd:choice>
              <xsd:group ref="Value"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="unitComponent">
          <xsd:complexType>
            <xsd:choice>
              <xsd:group ref="Unit"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
Now the EXPRESS SELECT types are translated into model groups.

<!-- group declarations (for SELECT data type definitions) -->
<xsd:group name="Unit">
  <xsd:choice>
    <xsd:element ref="DerivedUnit"/>
    <xsd:element ref="NamedUnit"/>
  </xsd:choice>
</xsd:group>

<!-- enumeration type declarations (for ENUMERATION data type definitions) -->
<xsd:simpleType name="DerivedUnitEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="angularvelocityunit"/>
    <xsd:enumeration value="compoundplaneangleunit"/>
    <xsd:enumeration value="dynamicviscosityunit"/>
    <xsd:enumeration value="heatfluxdensityunit"/>
    <xsd:enumeration value="integercountrateunit"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:enumeration value="isothermalmoisturecapacityunit"/>
<xsd:enumeration value="kinematicviscosityunit"/>
<xsd:enumeration value="massdensityunit"/>
<xsd:enumeration value="massflowrateunit"/>
<xsd:enumeration value="moisturediffusivityunit"/>
<xsd:enumeration value="molecularweightunit"/>
<xsd:enumeration value="specificheatcapacityunit"/>
<xsd:enumeration value="thermaladmittanceunit"/>
<xsd:enumeration value="thermalconductanceunit"/>
<xsd:enumeration value="thermalresistanceunit"/>
<xsd:enumeration value="thermaltransmittanceunit"/>
<xsd:enumeration value="vaporpermeabilityunit"/>
<xsd:enumeration value="volumetricflowrateunit"/>
<xsd:enumeration value="rotationalfrequencyunit"/>
<xsd:enumeration value="torqueunit"/>
<xsd:enumeration value="momentorinertiaunit"/>
<xsd:enumeration value="linearmomentunit"/>
<xsd:enumeration value="linearforceunit"/>
<xsd:enumeration value="planarforceunit"/>
<xsd:enumeration value="modulusofelasticityunit"/>
<xsd:enumeration value="shearmodulusunit"/>
<xsd:enumeration value="linearstiffnessunit"/>
<xsd:enumeration value="rotationalstiffnessunit"/>
<xsd:enumeration value="modulusofsubgradient reactionunit"/>
<xsd:enumeration value="accelerationunit"/>
<xsd:enumeration value="userdefined"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SIPrefix">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="exa"/>
    <xsd:enumeration value="peta"/>
    <xsd:enumeration value="tera"/>
    <xsd:enumeration value="giga"/>
    <xsd:enumeration value="mega"/>
    <xsd:enumeration value="kilo"/>
    <xsd:enumeration value="hecto"/>
    <xsd:enumeration value="deca"/>
    <xsd:enumeration value="deci"/>
    <xsd:enumeration value="centi"/>
    <xsd:enumeration value="milli"/>
    <xsd:enumeration value="micro"/>
    <xsd:enumeration value="nano"/>
    <xsd:enumeration value="pico"/>
    <xsd:enumeration value="femto"/>
    <xsd:enumeration value="atto"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SIUnitName">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="ampere"/>
    <xsd:enumeration value="becquerel"/>
    <xsd:enumeration value="candela"/>
    <xsd:enumeration value="cubic_metre"/>
    <xsd:enumeration value="degree_celsius"/>
    <xsd:enumeration value="farad"/>
    <xsd:enumeration value="gram"/>
    <xsd:enumeration value="hertz"/>
    <xsd:enumeration value="hecto"/>
    <xsd:enumeration value="hertzi"/>
    <xsd:enumeration value="kelvin"/>
    <xsd:enumeration value="lumen"/>
    <xsd:enumeration value="lux"/>
    <xsd:enumeration value="metre"/>
    <xsd:enumeration value="mole"/>
    <xsd:enumeration value="newton"/>
    <xsd:enumeration value="ohm"/>
    <xsd:enumeration value="pascal"/>
    <xsd:enumeration value="radian"/>
    <xsd:enumeration value="second"/>
    <xsd:enumeration value="siemens"/>
    <xsd:enumeration value="sievert"/>
    <xsd:enumeration value="square_metre"/>
    <xsd:enumeration value="steradian"/>
    <xsd:enumeration value="tesla"/>
    <xsd:enumeration value="volt"/>
    <xsd:enumeration value="watt"/>
    <xsd:enumeration value="weber"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="UnitEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="absorbeddoseunit"/>
    <xsd:enumeration value="amountofsubstanceunit"/>
    <xsd:enumeration value="areanunit"/>
    <xsd:enumeration value="doseequivvalentunit"/>
    <xsd:enumeration value="electriccapacitanceunit"/>
    <xsd:enumeration value="electricchargeunit"/>
    <xsd:enumeration value="electricconductanceunit"/>
    <xsd:enumeration value="electriccurrentunit"/>
    <xsd:enumeration value="electricresistanceunit"/>
    <xsd:enumeration value="electricvoltageunit"/>
    <xsd:enumeration value="energyunit"/>
    <xsd:enumeration value="forceunit"/>
    <xsd:enumeration value="frequencyunit"/>
  </xsd:restriction>
</xsd:simpleType>
Finally the defined type's are translated into simpleType's.

```
<!-- simple type declarations (for TYPE defined data type definitions) -->
<xsd:simpleType name="Label">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="Real">
  <xsd:restriction base="xsd:double"/>
</xsd:simpleType>
<xsd:simpleType name="Integer">
  <xsd:restriction base="xsd:long"/>
</xsd:simpleType>
```

For all schemas, generated by the ifcXML mapping, the following base declarations are given (omitting the configuration tags, highlighting default mapping).

```
<!-- standard declarations, elementType, simpleType, logical (for each translation) -->
<xsd:complexType name="element">
  <xsd:attribute use="optional" type="xsd:ID" name="id"/>
  <xsd:attribute use="optional" type="xsd:IDREF" name="href"/>
</xsd:complexType>
<xsd:attributeGroup name="configTags">
  <xsd:attribute default="choice" name="containmentType">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="containmentOnly"/>
        <xsd:enumeration value="referenceOnly"/>
        <xsd:enumeration value="choice"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute default="exclude" name="inverseAttributes">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="include"/>
        <xsd:enumeration value="exclude"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute default="exclude" name="derivedAttributes">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="include"/>
        <xsd:enumeration value="exclude"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute default="iso8601" name="timeDeclarations">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="iso8601"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```
4.1.4 SPF and XML instantiation examples

The following examples show instance data, following the original EXPRESS specification and the translated XML schema definitions.

4.1.4.1 Basic SI-units as global units

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

In IFC2x EXPRESS the SI units are defined as:
EXAMPLE An example where project’s global basic length, area and volume units are defined as SI units. Here all length measures are given as millimeter, all area measures as square meter and all volume measures as cubic meter.

```plaintext
#1= IFCPROJECT('fabcdghijklmnopqrst02',#7,'test project',$,$,$,$,(#20),#30);
/* global units */
#30=IFCUNITASSIGNMENT((#33, #34, #35));
#33=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
```

All measures within the geometric representations of IFC objects are given by measures with defined data types referring back to the global units, i.e. all lengths are given by the same unit (e.g. millimeter as in the example above). In general, all measure values within the whole project, that are given by IfcLengthMeasure, are now in millimeter, all IfcAreaMeasure are now in square meter, and all IfcVolumeMeasure are now in cubic meter.

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

```plaintext
#1= IFCPROJECT('fabcdghijklmnopqrst02',#7,'test project',$,$,$,$,(#20),#30);
#30=IFCUNITASSIGNMENT((#33, #34, #35, #36));
#33=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#36=IFCSIUNIT(*, .PLANEANGLEUNIT., $, .RADIAN.);
```

Translated into ifcXML for IFC2x the semantics remain the same. The translation into XML schema shows the following declaration of `<SIUnit>`. It follows the original type hierarchy and defines an `complexContent` by restriction, since the original IfcSIUnit contains redeclarations.

```
<UnitAssignment id="_1001">
  <units>
    <SIUnit name="millimetre" prefix="m" unitType="metre"/>
    <SIUnit name="square metre" unitType="metre"/>
    <SIUnit name="cubic metre" unitType="metre"/>
    <SIUnit name="radian" unitType="radian"/>
  </units>
</UnitAssignment>
```

The XML document to contain the global unit assignments (here in the context of the IfcMeasureResource) can be provided either based on containment (using embedded elements) or on reference, using empty elements referring to independent elements.

EXAMPLE The first example shows the declaration of millimetre, square metre and cubic metre as global unit assignments, using the containment approach.
EXAMPLE  The second example shows the declaration of millimetre, square metre, cubic metre and radian as global unit assignments, using the reference approach.

```xml
<SIUnit id="_1102">
  <unitType>lengthunit</unitType>
  <prefix>milli</prefix>
  <name>metre</name>
</SIUnit>

<SIUnit id="_1103">
  <unitType>areaunit</unitType>
  <name>square_metre</name>
</SIUnit>

<SIUnit id="_1104">
  <unitType>volumeunit</unitType>
  <name>cubic_metre</name>
</SIUnit>

<SIUnit id="_1105">
  <unitType>planeangleunit</unitType>
  <name>radian</name>
</SIUnit>
```

4.1.4.2 Conversion based units as global units

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

In IFC2x EXPRESS the conversion based units are defined as shown in Figure 9. Conversion based units apply to all non SI units, like imperial units.
Figure 9: IFC2x definition for conversion based unit

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

```xml
#1=IFCPROJECT('fabcdeghiklmnopqrst02', #7, 'test project', $, $, $, (#20), #30);
#30=IFCUNITASSIGNMENT((#33, #34, #35, #36));
#33=IFCSIUNIT(*, .LENGTHUNIT., .MILLI., .METRE.);
#34=IFCSIUNIT(*, .AREAUNIT., $, .SQUARE_METRE.);
#35=IFCSIUNIT(*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#36=IFCCONVERSIONBASEDUNIT(#40, .PLANEANGLEUNIT., 'DEGREE', #41);
#40=IFCDIMENSIONALEXPONENTS(0, 0, 0, 0, 0, 0, 0);
#41=IFCMESUREMENTWITHUNIT(IFCPLANEANGLEMEASURE(57.29577951308232), #50);
#50=IFCSIUNIT(*. .PLANEANGLEUNIT., $, .RADIAN.);
```

**EXAMPLE** where projects global basic length and area units are defined as imperial units (inches and square feet), which are further defined as conversion based units relative to SI units millimeter and square meter:

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

The translation into XML schema is shown in the next figure. The `<ConversionBasedUnit>` is given by `complexType` based on extension, as it does not redefine inherited attributes.
The XML document instances for the first example (the angle measure in degree) is given:

```xml
<UnitAssignment id="_1201">
  <units>
    <SIUnit id="_1202">
      <unitType>lengthunit</unitType>
      <prefix>milli</prefix>
      <name>metre</name>
    </SIUnit>
    <SIUnit id="_1203">
      <unitType>areaunit</unitType>
      <name>square_metre</name>
    </SIUnit>
    <SIUnit id="_1204">
      <unitType>volumeunit</unitType>
      <name>cubic_metre</name>
    </SIUnit>
    <ConversionUnit id="_1205">
      <dimensions>
        <DimensionalExponents>
          <lengthExponent>0</lengthExponent>
          <massExponent>0</massExponent>
          <timeExponent>0</timeExponent>
          <electricCurrentExponent>0</electricCurrentExponent>
          <thermodynamicTemperatureExponent>0</thermodynamicTemperatureExponent>
          <amountOfSubstanceExponent>0</amountOfSubstanceExponent>
          <luminousIntensityExponent>0</luminousIntensityExponent>
        </DimensionalExponents>
      </dimensions>
      <unitType>planeangleunit</unitType>
      <name>degree</name>
      <conversionFactor>
        <MeasureWithUnit>
          <valueComponent>
            <Real>57.29577951308232</Real>
          </valueComponent>
          <unitComponent>
            <SIUnit id="_1202"/>
          </unitComponent>
        </MeasureWithUnit>
      </conversionFactor>
    </ConversionUnit>
  </units>
</UnitAssignment>
```
The XML document instances for the second example (the length measure in inch, the area measure in square feet) is given:

```xml
<UnitAssignment id="_1301">
  <units>
    <ConversionBasedUnit id="_1302">
      <dimensions>
        <DimensionalExponents>
          <lengthExponent>1</lengthExponent>
          <massExponent>0</massExponent>
          <timeExponent>0</timeExponent>
          <electricCurrentExponent>0</electricCurrentExponent>
          <thermodynamicTemperatureExponent>0</thermodynamicTemperatureExponent>
          <amountOfSubstanceExponent>0</amountOfSubstanceExponent>
          <luminousIntensityExponent>0</luminousIntensityExponent>
        </DimensionalExponents>
      </dimensions>
      <unitType>lengthunit</unitType>
      <name>inch</name>
      <conversionFactor>
        <MeasureWithUnit>
          <valueComponent>
            <Real>25.40005</Real>
          </valueComponent>
          <unitComponent>
            <SIUnit>
              <unitType>lengthunit</unitType>
              <prefix>milli</prefix>
              <name>metre</name>
            </SIUnit>
          </unitComponent>
        </MeasureWithUnit>
      </conversionFactor>
    </ConversionBasedUnit>
    <ConversionBasedUnit id="_1303">
      <dimensions>
        <DimensionalExponents>
          <lengthExponent>2</lengthExponent>
          <massExponent>0</massExponent>
          <timeExponent>0</timeExponent>
          <electricCurrentExponent>0</electricCurrentExponent>
          <thermodynamicTemperatureExponent>0</thermodynamicTemperatureExponent>
          <amountOfSubstanceExponent>0</amountOfSubstanceExponent>
          <luminousIntensityExponent>0</luminousIntensityExponent>
        </DimensionalExponents>
      </dimensions>
      <unitType>areaunit</unitType>
      <name>square_feet</name>
      <conversionFactor>
        <MeasureWithUnit>
          <valueComponent>
            <Real>0.0929</Real>
          </valueComponent>
        </MeasureWithUnit>
      </conversionFactor>
    </ConversionBasedUnit>
  </units>
</UnitAssignment>
```
4.1.4.3 Derived units as global units

Derived units are units that are composed out of other units (either SI units or conversion based units). Note, that all units, that are derived from the 7 base SI units, but form part of the SI system in ISO 1000 are given as SI units, and not as derived units. Examples is Watt (W) for Nm/s, which is handled as an SI units, and not as a derived unit.

In IFC2x EXPRESS the conversion based units are defined as shown in Figure 11. Conversion based units apply to all non SI units, like imperial units.

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

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

The translation into XML schema is shown in the next figure.
The XML document instances for the example (specific heat capacity) is given again using containment of embedded elements:

```xml
[UnitAssignment id="_1401">
  <units>
    <DerivedUnit id="_1402">
      <elements>
        <DerivedUnitElement>
          <unit>
            <SIUnit>
              <unitType>energyunit</unitType>
              <name>joule</name>
            </SIUnit>
            <exponent>1</exponent>
          </unit>
        </DerivedUnitElement>
        <DerivedUnitElement>
          <unit>
            <SIUnit>
              <unitType>massunit</unitType>
              <prefix>kilo</prefix>
              <name>gram</name>
            </SIUnit>
            <exponent>-1</exponent>
          </unit>
        </DerivedUnitElement>
        <DerivedUnitElement>
          <unit>
            <SIUnit>
              <unitType>thermodynamictemperatureunit</unitType>
              <name>kelvin</name>
            </SIUnit>
            <exponent>-1</exponent>
          </unit>
        </DerivedUnitElement>
      </elements>
      <unitType>specificheatcapacityunit</unitType>
    </DerivedUnit>
  </units>
</UnitAssignment>
```