Support for reliable data exchange in ISO 12006-3

Value types, machine readable units and external schema representations

Table of Contents

INTRODUCTION

Based on the suggested topics outlined in document ISO-TC59-SC13-WG6_N0233 and our own investigation of the ISO 12006-3 standard we have focused our effort towards improved definitions of values to provide reliable data exchange definitions which can be used to inform data and data templates in IFC and external automated systems in general. Current implementation is focused on the human communication which is sufficient for the tasks where common agreement on the concept is a key. But it lacks reliable machine readable definitions of units and reliable numeric and logical values encoding which makes it hard or impossible to use those agreed multilingual concepts to inform external automated systems or IFC data or data templates.

UNITS

IFC (ISO 16739) already contains data schema which describes units in reliable way and it also uses EXPRESS modelling language and diagrams for data modelling. These data definitions are related to SI units and allow to define any unit based on those. Interoperability with IFC is a key in context of building information modelling related standards. While N0233 document only suggests introduction of dimensional exponents we see this as insufficient because dimensional exponents only define kind of a measure (length, area, acceleration), but not the unit itself (meter, meter squared, meters per second etc.).

We recognize that because IFC and ISO 12006-3 both use EXPRESS modelling language it would be possible to reference selected data entities from IFC in the ISO 12006-3 directly, making them part of the main schema. Part of the IFC schema describing units is almost self-contained (without dependencies on other parts of the schema). But it is not independent completely so this approach would indirectly import number of entity types and defined types which are not necessary for the scope of ISO 12006- 3.

Because of that we decided to use another approach where we isolated part of the IFC data schema necessary to describe units in a full depth, made minor modifications to make it completely self-contained and used ISO 12006-3 naming convention to make those part of the core schema.

After the initial implementation we found that rooted units. For example $kg/(m^2 \cdot \sqrt{h}) = kg \cdot m^{-2} \cdot h^{-\frac{1}{2}} = kg \cdot m^{-2} \cdot h^{-0.5}$ can't be expressed in IFC because it only allows integral exponents. Because IFC is based on ISO 10303-41 we made further refinements of the proposed schema so that it is closer to simpler definitions in ISO 10303-41 in some aspects including real numbers for dimensional exponents. Where IFC requires duplicated declarations we removed this potential for ambiguities. Where IFC handles names for various units we removed this because naming is the

main goal of xtdUnit which provides data infrastructure to describe relations of concepts and multilingual names for the shared concept.

This common understanding should be based on an unambiguous definition of the unit as we propose. Resulting schema is simpler compared to IFC but is still compatible except for the rooted units as mentioned above. In other cases IFC values and additional enumerations can be directly inferred from the definition.

VALUES

Values are currently only stored as a text in ISO 12006-3. This is sufficient for human communication over concepts but can't be used reliably to exchange numeric of logical data. EXPRESS schema already contains native support for other base data types and xtdValue should be able to contain these as well. Using these types brings reliable and well defined rules for value encoding when stored as STEP21, XML or other physical format. Together with machine readable units this provides base support for reliable data exchange between IFC and ISO 12006-3.

When investigating this part of the schema we also realised that xtdValue is used to define both the value or the value constraint. This allows for the level of ambiguity which might make it hard even for humans to get to common understanding of the actual meaning. Based on this observation we make a suggestion to separate those two concepts and to introduce xtdConstraint entity type which can be applied to property in a similar way to xtdValue. As a result, property might have a value and/or constraint (or more of them).

For scenarios like product data templates constraints would be a lot more appropriate to describe actual concepts in the schema with good level of common understanding. Several new relations are introduced in our suggestion to support similar relations between xtdObject, xtdProperty and xtdConstraint as is already available for xtdProperty.

EXTERNAL SCHEMAS

If ISO 12006-3 is used to inform structure of external data than it is likely to represent data concepts present in external schemas. IFC is one of a particular interest as a widely used and adopted standard. But suggested approach is usable for any external system. It introduces new entity types which describe both the schema and external entity. External systems can use this information to identify the mapping reliably. Any xtdObject can have a mapping to external schema object. External object can be described by identifier and sub-identifier. This allows for more detailed description of the object. In case of IFC this can be used to describe entity type by identifier and predefined type as sub-identifier. Other external automated systems might use it for other fine grained differentiation.

SUGGESTED CHANGES IN THE SCHEMA

Following text contains suggested modifications in the schema with comments containing some of the reasoning explained above. Complete working schema in a pure text is provided alongside with this document.

To make development of the schema easier we developed a SW toolkit which makes it possible to generate implementation of the schema and to test its functionality. This toolkit is written in .NET (platform independent .NET Core) and is freely available on GitHub: https://github.com/martin1cerny/ISO12006-3

This toolkit was used to produce examples at the end of this document.

New entities, types and functions

Semantic model for constraints

```
(*
  New area of datastructures is introduced to describe requirements related to
properties.
  Current version was combining both declarations of values and declarations of
requirements.
  This is split into two distinct concepts now. Constraints can be combined using logical
  operators forming complex trees if necessary. Constraints can limit valid domain of
values
  as well as a domain of valid units, potentially combining both.
*)
ENTITY xtdConstraint
  ABSTRACT SUPERTYPE OF (ONEOF(xtdMeasureConstraint, xtdUnitConstraint,
xtdLogicalConstraint))
  SUBTYPE OF(xtdObject);
END_ENTITY;

ENTITY xtdMeasureConstraint
  SUBTYPE OF(xtdConstraint);
  ConstraintType: xtdConstraintTypeEnum;
  ConstraintValues: SET [1:?] OF xtdValueType;
  ValuesUnit: OPTIONAL xtdUnit;
END_ENTITY;

ENTITY xtdMeasureIntervalConstraint
  SUBTYPE OF(xtdConstraint);
  UpperBoundary: xtdReal;
  LowerBoundary: xtdReal;
  IncludingUpperBoundary: BOOLEAN;
  IncludingLowerBoundary: BOOLEAN;
  ValuesUnit: OPTIONAL xtdUnit;
END_ENTITY;

ENTITY xtdUnitConstraint
  SUBTYPE OF(xtdConstraint);
  ConstraintUnit: xtdUnit;
END_ENTITY;

ENTITY xtdLogicalConstraint
  SUBTYPE OF(xtdConstraint);
  LogicalOperator: xtdLogicalOperatorEnum;
  Constraints: SET [1: ?] OF xtdConstraint;
END_ENTITY;


ENTITY xtdRelAssignsConstraint
```

```
  SUBTYPE OF(xtdRelationship);
    RelatingProperty : xtdProperty;
    RelatedConstraint : xtdConstraint;
    MethodOfInterpretation : OPTIONAL xtdName;
END_ENTITY;

ENTITY xtdRelAssignsPropertyWithConstraint
  SUBTYPE OF(xtdRelationship);
    RelatedProperty : xtdProperty;
    RelatingObject : xtdObject;
    RelatedConstraint :  xtdConstraint;
END_ENTITY;

TYPE xtdConstraintTypeEnum = ENUMERATION OF
       (
       GREATERTHAN,
       GREATERTHANOREQUALTO,
       LESSTHAN,
       LESSTHANOREQUALTO,
       EQUALTO,
       NOTEQUALTO,
       INCLUDES,
       NOTINCLUDES,
       INCLUDEDIN,
       NOTINCLUDEDIN
       );
END_TYPE;

TYPE xtdLogicalOperatorEnum = ENUMERATION OF
       (
       OP_AND,
       OP_OR,
       OP_XOR
       );
END_TYPE;
```

Specific defined value types

```
(*
  In addition to existing defined types (xtdLabel, xtdText and xtdDate) new defined types
  are introduced to cover base types used in data modelling and as base EXPRESS data
types.
  These make exchange of numeric values reliable because they make it possible to rely on
  encoding of numeric values. This is mainly used for xtdValue where all these types are
  made available through a new select type xtdValueType.

*)
TYPE xtdNumber = NUMBER;
END_TYPE;

TYPE xtdInteger = INTEGER;
END_TYPE;

TYPE xtdBoolean = BOOLEAN;
END_TYPE;

TYPE xtdLogical = LOGICAL;
END_TYPE;

TYPE xtdReal = REAL;
END_TYPE;

TYPE xtdPercent = REAL;
END_TYPE;

TYPE xtdValueType = SELECT
       (xtdLabel,
       xtdText,
       xtdReal,
       xtdNumber,
```

```
       xtdInteger,
       xtdBoolean,
       xtdLogical,
       xtdDate,
       xtdPercent);
END_TYPE;
```

Mapping between IFD concepts and external schemas

```
(*
  Following new entity types make it possible to describe relations between
  concepts expressed in IFD and their representation in external schemas like IFC
  or others. This is essential for automated processes where IFD data is informing
  content and/or structure of data defined using external schemas.
*)

ENTITY xtdRelMapping
  SUBTYPE OF(xtdRelationship);
    RelatingObject : xtdObject;
    RelatedExternalObject : xtdExternalObject;
END_ENTITY;

ENTITY xtdExternalObject;
    ExternalIdentifier : xtdLabel;
    ExternalSubIdentifier : OPTIONAL xtdLabel;
      ExternalSchema: xtdExternalSchema;
END_ENTITY;

ENTITY xtdExternalSchema;
    Name: OPTIONAL xtdLabel;
      Description: OPTIONAL xtdText;
      Identifier: xtdLabel;
      Version: xtdLabel;
      Location: OPTIONAL xtdLabel;
END_ENTITY;
```

Machine readable units

```
(*
  While ISO 12006-3 is perfectly usable to model machine readable data
  it didn't have a data structures which could be used to describe units
  in a machine readable way. The concept of xtdUnit seems to be targeting
  communication between people where it is more important to agree on names
  and definitions. But once those units are used to describe values and those
  values are to be used for validation or to inform a data content in other
  automated systems this is not sufficient. Following structures are inspired
  by definitions used in IFC and ISO 10303-41. These are simplified compared to
  definitions in IFC but are fully compatible with them except for the very rare
  cases of rooted units (like "hour to the power of one half"). Following definitions
  don't allow any ambiguity and can secure reliable exchange of information.
*)

(* This select type should be used as a mandatory attribute of xtdUnit which provides
   machine readable definition where people can than agree on multilingual names,
   descriptions and relations *)
TYPE xtdUnitDefinition = SELECT
       (xtdDerivedUnit
       ,xtdNamedUnit);
END_TYPE;

(* This unit doesn't have any 'Name' attribute as that should be defined in xtdUnit where
this
   is used as the machine readable definition *)
ENTITY xtdDerivedUnit;
       Elements : SET [1:?] OF xtdDerivedUnitElement;
 DERIVE
       Dimensions : xtdDimensionalExponents := xtdDeriveDimensionalExponents(Elements);
 WHERE
```

```
       WR1 : (SIZEOF (Elements) > 1) OR ((SIZEOF (Elements) = 1) AND
(Elements[1].Exponent <> 1 ));
END_ENTITY;

(* Element of the derived unit *)
ENTITY xtdDerivedUnitElement;
       Unit : xtdNamedUnit;
       Exponent : REAL;
END_ENTITY;

ENTITY xtdNamedUnit
 ABSTRACT SUPERTYPE OF (ONEOF
    (xtdContextDependentUnit
    ,xtdConversionBasedUnit
    ,xtdSIUnit));
 DERIVE
       Dimensions : xtdDimensionalExponents := xtdDimensionsForNamedUnit (SELF);
END_ENTITY;

(* Context dependent units are not related to SI units and are supposed to have
dimensionality = 1
   The name should be the agreed symbol for this unit. Name should be defined in xtdUnit
using this
   as the definition.
*)
ENTITY xtdContextDependentUnit
 SUBTYPE OF (xtdNamedUnit);
       Name : xtdLabel;
END_ENTITY;

(* Conversion based units are litre, hour, minute, feet and others which are
   related to SI units by the conversion factor and/or offset
(xtdConversionBasedUnitWithOffset) *)
ENTITY xtdConversionBasedUnit
 SUPERTYPE OF (ONEOF
    (xtdConversionBasedUnitWithOffset))
 SUBTYPE OF (xtdNamedUnit);
       Name : xtdLabel;
       ConversionFactor : xtdNumber;
       BaseUnit: xtdUnitDefinition;
END_ENTITY;

ENTITY xtdConversionBasedUnitWithOffset
 SUBTYPE OF (xtdConversionBasedUnit);
       ConversionOffset : xtdReal;
END_ENTITY;

(* Both prefix and name are fixed enumerations *)
ENTITY xtdSIUnit
 SUBTYPE OF (xtdNamedUnit);
       Prefix : OPTIONAL xtdSIPrefix;
       Name : xtdSIUnitName;
END_ENTITY;

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

```
END_TYPE;

(* Compared to IFC this doesn't contain .CUBIC_METRE. and .SQUARE_METRE. because
   these are ambiguous when combined with the prefix *)
TYPE xtdSIUnitName = ENUMERATION OF
       (AMPERE
       ,BECQUEREL
       ,CANDELA
       ,COULOMB
       ,DEGREE_CELSIUS
       ,FARAD
       ,GRAM
       ,GRAY
       ,HENRY
       ,HERTZ
       ,JOULE
       ,KELVIN
       ,LUMEN
       ,LUX
       ,METRE
       ,MOLE
       ,NEWTON
       ,OHM
       ,PASCAL
       ,RADIAN
       ,SECOND
       ,SIEMENS
       ,SIEVERT
       ,STERADIAN
       ,TESLA
       ,VOLT
       ,WATT
       ,WEBER);
END_TYPE;

(* Dimensional exponents are defined as INTEGER in IFC but as REAL in ISO_10303-41.
   INTEGER values wouldn't allow to express rare cases of rooted unit (like an hour to
the power of one half)*)
ENTITY xtdDimensionalExponents;
       LengthExponent : REAL;
       MassExponent : REAL;
       TimeExponent : REAL;
       ElectricCurrentExponent : REAL;
       ThermodynamicTemperatureExponent : REAL;
       AmountOfSubstanceExponent : REAL;
       LuminousIntensityExponent : REAL;
END_ENTITY;

(*
          Functions returning correct dimensional exponents and unit types of units
   ------------------------------------------------------------
*)

FUNCTION xtdDimensionsForNamedUnit
  (unit : xtdNamedUnit )      : xtdDimensionalExponents;
    IF 'XTDSIUNIT' IN TYPEOF(unit) THEN
        RETURN (xtdDimensionsForSiUnit(unit));
    END_IF;
       IF 'XTDCONVERSIONBASEDUNIT' IN TYPEOF(unit) THEN
        RETURN (unit\xtdConversionBasedUnit.BaseUnit.Dimensions);
    END_IF;
       IF 'XTDCONTEXTDEPENDENTUNIT' IN TYPEOF(unit) THEN
        RETURN (xtdDimensionalExponents(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
    END_IF;
END_FUNCTION;


FUNCTION xtdDimensionsForSiUnit
(n : xtdSiUnitName )      : xtdDimensionalExponents;
  CASE n OF
    METRE          : RETURN (xtdDimensionalExponents
```

```
                                (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
    GRAM            : RETURN (xtdDimensionalExponents
                                (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0));
    SECOND          : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0));
    AMPERE          : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0));
    KELVIN          : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0));
    MOLE            : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0));
    CANDELA         : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
    RADIAN          : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
    STERADIAN       : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
    HERTZ           : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0));
    NEWTON          : RETURN (xtdDimensionalExponents
                                (1.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
    PASCAL          : RETURN (xtdDimensionalExponents
                                (-1.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
    JOULE           : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
    WATT            : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -3.0, 0.0, 0.0, 0.0, 0.0));
    COULOMB         : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0));
    VOLT            : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -3.0, -1.0, 0.0, 0.0, 0.0));
    FARAD           : RETURN (xtdDimensionalExponents
                                (-2.0, -1.0, 4.0.0, 2.0, 0.0, 0.0, 0.0));
    OHM             : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -3.0, -2.0, 0.0, 0.0, 0.0));
    SIEMENS         : RETURN (xtdDimensionalExponents
                                (-2.0, -1.0, 3.0, 2.0, 0.0, 0.0, 0.0));
    WEBER           : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -2.0, -1.0, 0.0, 0.0, 0.0));
    TESLA           : RETURN (xtdDimensionalExponents
                                (0.0, 1.0, -2.0, -1.0, 0.0, 0.0, 0.0));
    HENRY           : RETURN (xtdDimensionalExponents
                                (2.0, 1.0, -2.0, -2.0, 0.0, 0.0, 0.0));
    DEGREE_CELSIUS : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0));
    LUMEN           : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
    LUX             : RETURN (xtdDimensionalExponents
                                (-2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
    BECQUEREL       : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0));
    GRAY            : RETURN (xtdDimensionalExponents
                                (2.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0));
    SIEVERT         : RETURN (xtdDimensionalExponents
                                (2.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0));
    OTHERWISE       : RETURN (xtdDimensionalExponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
  END_CASE;
END_FUNCTION;
```

Modified entities

```
(*

  Unit is modified to contain machine readable definition of the
  unit related to SI units. These definitions are 1:1 with IFC definitions
  This makes it possible to interpres related values unambiguously.
*)
ENTITY xtdUnit
```

```
  SUBTYPE OF(xtdObject);
  Definition: xtdUnitDefinition;
END_ENTITY;

(*
   Value is modified so that numeric and logical or boolean values
   are encoded using STEP21 definitions making the value reliable for
   automated data exchange. xtdValueTypeEnum is not needed as a result.

   Attribute defining constraints (ValueRole) removed. New class modelling concept of
   a value constraint is introduced which can be used to describe requirements
   for a property.

   These constrain can form logical trees with logical predicates and can put
   requirements both on values and units allowing flexible combinations.
*)
ENTITY xtdValue
  SUBTYPE OF(xtdObject);
    LowerTolerance : OPTIONAL xtdValueType;
    NominalValue : OPTIONAL xtdValueType;
    UpperTolerance : OPTIONAL xtdValueType;
    ToleranceType : OPTIONAL xtdToleranceTypeEnum;
END_ENTITY;


(*
  RelatedUnit is added to give a scale to the values
*)
ENTITY xtdRelAssignsPropertyWithValues
  SUBTYPE OF(xtdRelationship);
    RelatedProperty : xtdProperty;
    RelatingObject : xtdObject;
    RelatedValues : LIST [1:?] OF UNIQUE xtdValue;
       RelatedUnit: OPTIONAL xtdUnit;
END_ENTITY;
```

Deleted entities

```
(*
// xtdValueTypeEnum should be removed because real typing should be used. This will
ensure
// proper encoding of numeric calues
TYPE xtdValueTypeEnum = ENUMERATION OF
  (XTDSTRING,
  XTDNUMBER,
  XTDINTEGER,
  XTDREAL,
  XTDBOOLEAN,
  XTDLOGICAL);
END_TYPE;

// Values and constraints are modeled as two different concepts
TYPE xtdValueRoleEnum = ENUMERATION OF
  (NOMINAL,
  MAXIMUM,
  MINIMUM);
END_TYPE;
*)
```

EXAMPLE A: PROPERTY VALUES AND UNITS

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION ((''), '2;1');
FILE_NAME ('', '2018-12-20T12:32:13', (''), (''), 'Processor version 5.0.0.0',
'Xbim.Common', '');
FILE_SCHEMA (('ISO_12006_3_VERSION_4'));
ENDSEC;
DATA;

/**
* Definition of the language
* XTDLANGUAGE=(LanguageNameInEnglish, LanguageNameInSelf, Comments, UniqueID)*/
#1=XTDLANGUAGE('English',$,$,'2_E38JsQD2BA5Bsv_cgIBi');

/**
* New entity types introduced to describe units. This is a simple SI unit.
* XTDSIUNIT=(Prefix, Name)*/
#2=XTDSIUNIT(.MILLI.,.METRE.);

/**
* Speed is derived from length unit of metres and time unit of seconds with appropriate
exponents
* XTDDERIVEDUNIT=(Elements)*/
#3=XTDDERIVEDUNIT((#4,#6));
#4=XTDDERIVEDUNITELEMENT(#5,1.);
#5=XTDSIUNIT($,.METRE.);
#6=XTDDERIVEDUNITELEMENT(#7,-1.);
#7=XTDSIUNIT($,.SECOND.);

/**
* xtdUnit is extended with 'Definition' which is a machine readable expression for the
unit.
* This makes it possible to external autometed systems to work with the data and
definitions.
* XTDUNIT=(VersionDate, VersionID, UniqueID, Descriptions, Names, Definition)*/
#8=XTDUNIT('2018-12-20T12:32:13','2.1','1ygnYWt2LAlAVFb3V71oxD',$,(#9),#2);
#9=XTDNAME(#1,'0P2016Tf5EafCK9VZOEDxR','Millimetre');
#10=XTDUNIT('2018-12-20T12:32:13','2.1','3DP$VUay98dhrX60LIeg6L',$,(#11),#3);
#11=XTDNAME(#1,'14wnDdiy15sPeRO_z8IrtC','Metres per second');

/**
* Subject of the specification
* XTDSUBJECT=(VersionDate, VersionID, UniqueID, Descriptions, Names)*/
#12=XTDSUBJECT('2018-12-20T12:32:13','2.1','3MyjV43s557vJmwfbzadik',(#14),(#13));
#13=XTDNAME(#1,'1COwmbAnXCy88_JRZTqV9d','The Door');
#14=XTDDESCRIPTION(#1,'3W6K4JW_9EZu8rOQdNDGYn','Example of the door');

/**
* Property can either be related to subject with value or constraint or both
* XTDPROPERTY=(VersionDate, VersionID, UniqueID, Descriptions, Names)*/
#15=XTDPROPERTY('2018-12-20T12:32:13','2.1','0mwakRjBX6WBBNRlqWbh3w',(#17),(#16));
#16=XTDNAME(#1,'1ygBxIIkP1TuGv$1yO18LA','Width');
#17=XTDDESCRIPTION(#1,'1Em1B5iKfA3P6XF3uZ0nou','Clear widt of the door');
#18=XTDPROPERTY('2018-12-20T12:32:13','2.1','0gLhYHtmX3EhJ2bLR5IVAb',(#20),(#19));
#19=XTDNAME(#1,'2Ca8SPy5j0ERzonvlMikVo','Quality');
#20=XTDDESCRIPTION(#1,'2kyR9QuAXA_QnZJVvChGoq','Quality grading');
#21=XTDPROPERTY('2018-12-20T12:32:13','2.1','1HrmF2JKH9Ig0Db1hwQJO_',(#23),(#22));
#22=XTDNAME(#1,'1MlWSD7czCCvFrCl$8zG8_','Opening Speed');
#23=XTDDESCRIPTION(#1,'3bX7g6IGn8HhczLG6Vp6wr','Speed of automatic opening of the door');

/**
* This measure has value reliably encoded as a real number and has machine readable units
* XTDMEASUREWITHUNIT=(VersionDate, VersionID, UniqueID, Descriptions, Names,
UnitComponent, ValueDomain)*/
#24=XTDMEASUREWITHUNIT('2018-12-20T12:32:13','2.1','2zxhCF2DP6HB0JPA2kgZVG',(#26),
(#25),#8,(#27));
#25=XTDNAME(#1,'0aeRAZeTnCVvO_APCuw4rj','Width measure');
#26=XTDDESCRIPTION(#1,'2X1IQRzR94hwzN_hj0swJT','Width of the door');
```

```
#27=XTDVALUE('2018-12-20T12:32:13','2.1','3ZsFELQWj7PBR3$RQ0e2yM',$,(),$,XTDREAL(800.),$,
$);

/**
* This measure represents the grade as an enumeration value
* XTDMEASUREWITHUNIT=(VersionDate, VersionID, UniqueID, Descriptions, Names,
UnitComponent, ValueDomain)*/
#28=XTDMEASUREWITHUNIT('2018-12-20T12:32:13','2.1','06ENt5V8rBmB7y7AApaWY5',(#30),
(#29),#8,(#31));
#29=XTDNAME(#1,'2NrhiLsbHBAfwOG69J6YkA','Quality grade');
#30=XTDDESCRIPTION(#1,'2m_WnThJz0ARhbHbsQ1f0e','Quality must be expressed as one of the
values [A,B,C,D,E]');
#31=XTDVALUE('2018-12-20T12:32:13','2.1','3GL83duq1DnesaJckYbc0O',$,(),$,XTDLABEL('C'),$,
$);

/**
* This measure represents the speed with well defined units
* XTDMEASUREWITHUNIT=(VersionDate, VersionID, UniqueID, Descriptions, Names,
UnitComponent, ValueDomain)*/
#32=XTDMEASUREWITHUNIT('2018-12-20T12:32:13','2.1','3XzpI518DEoRmX0wd7Lt6N',(#34),
(#33),#10,(#35));
#33=XTDNAME(#1,'25BgA9Uhr6ffiBQH2ODNq5','Speed');
#34=XTDDESCRIPTION(#1,'0ZAohUo7vFsOFKMaKfH0P8','Speed under normal conditions');
#35=XTDVALUE('2018-12-20T12:32:13','2.1','0KmEdb1V17U8p2Kp9BHlW7',$,(),$,XTDREAL(1.3456),
$,$);

/**
* Relations expressing measures of the properties with units for certain subject
* XTDRELASSIGNSPROPERTYWITHVALUES=(VersionDate, VersionID, UniqueID, Descriptions, Names,
ViewSelector, RelatedProperty, RelatingObject, RelatedValues, RelatedUnit)*/
#36=XTDRELASSIGNSPROPERTYWITHVALUES('2018-12-20T12:32:13','2.1','1gU_mMMoT73fExW7UWRZyd',
$,(),$,#15,#12,(),#8);
#37=XTDRELASSIGNSPROPERTYWITHVALUES('2018-12-20T12:32:13','2.1','3Ae$vqdNX7Jw6F_b0aDiWw',
$,(),$,#18,#12,(),$);
#38=XTDRELASSIGNSPROPERTYWITHVALUES('2018-12-20T12:32:13','2.1','0we2ORm711X9I8TMZTZckM',
$,(),$,#21,#12,(),#10);
ENDSEC;
END-ISO-10303-21;
```

EXAMPLE B: SUBJECT WITH CONSTRAINTS ON PROPERTIES

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION ((''), '2;1');
FILE_NAME ('', '2018-12-20T12:32:13', (''), (''), 'Processor version 5.0.0.0',
'Xbim.Common', '');
FILE_SCHEMA (('ISO_12006_3_VERSION_4'));
ENDSEC;
DATA;

/**
* Definition of the language
* XTDLANGUAGE=(LanguageNameInEnglish, LanguageNameInSelf, Comments, UniqueID)*/
#1=XTDLANGUAGE('English',$,$,'1InZqrgDr20eEfjN_kfyot');

/**
* New entity types introduced to describe units. This is a simple SI unit.
* XTDSIUNIT=(Prefix, Name)*/
#2=XTDSIUNIT(.MILLI.,.METRE.);

/**
* xtdUnit is extended with 'Definition' which is a machine readable expression for the
unit.
* This makes it possible to external autometed systems to work with the data and
definitions.
* XTDUNIT=(VersionDate, VersionID, UniqueID, Descriptions, Names, Definition)*/
#3=XTDUNIT('2018-12-20T12:32:13','2.1','2YPicS45j7DeYbHUd8NCvP',$,(#4),#2);
#4=XTDNAME(#1,'2tzSgEszz1duDjYQSItbXN','Millimetre');

/**
* Subject of the specification
* XTDSUBJECT=(VersionDate, VersionID, UniqueID, Descriptions, Names)*/
#5=XTDSUBJECT('2018-12-20T12:32:13','2.1','0MFIwdTsT3DQSd4ZIqgp8k',(#7),(#6));
#6=XTDNAME(#1,'3FM23I$I19hOwTNaLFdHzK','The Door');
#7=XTDDESCRIPTION(#1,'3$nYU_fBD6xAnfMQNt2NiK','Example of the door');

/**
* Property can either be related to subject with value or constraint or both
* XTDPROPERTY=(VersionDate, VersionID, UniqueID, Descriptions, Names)*/
#8=XTDPROPERTY('2018-12-20T12:32:13','2.1','0bvMszyF197fFKx5_KyTN7',(#10),(#9));
#9=XTDNAME(#1,'2VNHHxr7nCPetIabf1AHdF','Width');
#10=XTDDESCRIPTION(#1,'3bjDbpzdH2$gxAAl$i4aHF','Clear width of the door');
#11=XTDPROPERTY('2018-12-20T12:32:13','2.1','1lBbpcBE58lgWuM3n1oxnR',(#13),(#12));
#12=XTDNAME(#1,'3Ph0EcfJbD2OI$V9TofaN2','Quality');
#13=XTDDESCRIPTION(#1,'2COHzQ0Mf3Auy812K5Ds2e','Quality grading');

/**
* This constraint expresses requirement for the value assignable to a property
* XTDMEASURECONSTRAINT=(VersionDate, VersionID, UniqueID, Descriptions, Names,
ConstraintType, ConstraintValues, ValuesUnit)*/
#14=XTDMEASURECONSTRAINT('2018-12-20T12:32:13','2.1','20Sz6Cai99JfiAZmnyJqv0',(#16),
(#15),.LESSTHANOREQUALTO.,(XTDREAL(800.)),#3);
#15=XTDNAME(#1,'2RNifhVHL3KPtxypZaHnsP','Width constraint');
#16=XTDDESCRIPTION(#1,'0Bn_OK$f9EzuVxS3URJqmG','Width of the doot must be less than or
equal to 800mm');

/**
* This constraint expresses requirement for the value to exist in the list of allowed
values (enumeration)
* XTDMEASURECONSTRAINT=(VersionDate, VersionID, UniqueID, Descriptions, Names,
ConstraintType, ConstraintValues, ValuesUnit)*/
#17=XTDMEASURECONSTRAINT('2018-12-20T12:32:13','2.1','3OC5tLJOb2U8fGsrZ7lIJn',(#19),
(#18),.INCLUDEDIN.,
(XTDLABEL('A'),XTDLABEL('B'),XTDLABEL('C'),XTDLABEL('D'),XTDLABEL('E')),#3);
#18=XTDNAME(#1,'0Q9t3RqIf1oveXlo$tTesx','Quality grade constraint');
#19=XTDDESCRIPTION(#1,'1ZWOOSidH1sPZ2i3H76CkF','Quality must be expressed as one of the
values');

/**
```

```
* Relation expressing constrained property for the subject
* XTDRELASSIGNSPROPERTYWITHCONSTRAINT=(VersionDate, VersionID, UniqueID, Descriptions,
Names, ViewSelector, RelatedProperty, RelatingObject, RelatedConstraint)*/
#20=XTDRELASSIGNSPROPERTYWITHCONSTRAINT('2018-12-
20T12:32:13','2.1','1802kSPBXC2OST1GUP7gke',$,(),$,#8,#5,#14);
#21=XTDRELASSIGNSPROPERTYWITHCONSTRAINT('2018-12-
20T12:32:13','2.1','2y_IcPNXP73OsGJCbQmCyf',$,(),$,#11,#5,#17);
ENDSEC;
END-ISO-10303-21;
```

EXAMPLE C: MAPPING TO EXTERNAL SCHEMA OBJECTS

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION ((''), '2;1');
FILE_NAME ('', '2018-12-20T12:32:13', (''), (''), 'Processor version 5.0.0.0',
'Xbim.Common', '');
FILE_SCHEMA (('ISO_12006_3_VERSION_4'));
ENDSEC;
DATA;

/**
* Definition of the language
* XTDLANGUAGE=(LanguageNameInEnglish, LanguageNameInSelf, Comments, UniqueID)*/
#1=XTDLANGUAGE('English',$,$,'2wRFNVuEb9KRmbFzIIyJW$');

/**
* Subject of the specification
* XTDSUBJECT=(VersionDate, VersionID, UniqueID, Descriptions, Names)*/
#2=XTDSUBJECT('2018-12-20T12:32:13','2.1','01kfXsJwH0tPhxmxnV$YDn',(#4),(#3));
#3=XTDNAME(#1,'3DLWlHBbX1Ehp$H1e2w74O','The Door');
#4=XTDDESCRIPTION(#1,'2vst_cwtL5dA_uo3I824Eb','Example of the door');
#5=XTDSUBJECT('2018-12-20T12:32:13','2.1','0FuiQR2aP0Kwpx50pH84sc',(#7),(#6));
#6=XTDNAME(#1,'03NjHJ5zfCVeoVh1Mbl9ew','The Floor');
#7=XTDDESCRIPTION(#1,'2G8T76p29BjOTjXIziBl6z','Example of the floor');

/**
* Information identifying external schema so that external automation systems can
identify relevant mappings
* XTDEXTERNALSCHEMA=(Name, Description, Identifier, Version, Location)*/
#8=XTDEXTERNALSCHEMA('IFC','Industrial Foundation Classes','IFC4','IFC4
ADD2','http://www.buildingsmart-tech.org/ifc/IFC4/Add2/IFC4_ADD2.exp');

/**
* External objects identified by their identifiers, sub-identifiers and the schema
* XTDEXTERNALOBJECT=(ExternalIdentifier, ExternalSubIdentifier, ExternalSchema)*/
#9=XTDEXTERNALOBJECT('IfcDoor','DOOR',#8);
#10=XTDEXTERNALOBJECT('IfcSlab','FLOOR',#8);

/**
* Relation defining external objects in external schemas
* XTDRELMAPPING=(VersionDate, VersionID, UniqueID, Descriptions, Names, ViewSelector,
RelatingObject, RelatedExternalObject)*/
#11=XTDRELMAPPING('2018-12-20T12:32:13','2.1','1LZSnD2gL8FwyOoeKvUZYD',$,(),$,#2,#9);
#12=XTDRELMAPPING('2018-12-20T12:32:13','2.1','2xFW$pw2nE9xe3AGAf_0gr',$,(),$,#5,#10);
ENDSEC;
END-ISO-10303-21;
```

EXAMPLE D: COMPLEX UNITS EXAMPLE

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (('''), '2;1');
FILE_NAME ('', '2018-12-20T12:32:13', (''), (''), 'Processor version 5.0.0.0',
'Xbim.Common', '');
FILE_SCHEMA (('ISO_12006_3_VERSION_4'));
ENDSEC;
DATA;

/**
* Litre is m³/1000
* XTDCONVERSIONBASEDUNIT=(Name, ConversionFactor, BaseUnit)*/
#1=XTDCONVERSIONBASEDUNIT('litre',0.001,#2);
#2=XTDDERIVEDUNIT((#3));
#3=XTDDERIVEDUNITELEMENT(#4,3.);
#4=XTDSIUNIT($,.METRE.);

/**
* litre per metre squared second   l / (m²·s)
* Dimensional exponents: [1.00 0.00 -1.00 0.00 0.00 0.00 0.00]
* XTDDERIVEDUNIT=(Elements)*/
#5=XTDDERIVEDUNIT((#6,#7,#9));
#6=XTDDERIVEDUNITELEMENT(#1,1.);
#7=XTDDERIVEDUNITELEMENT(#8,-2.);
#8=XTDSIUNIT($,.METRE.);
#9=XTDDERIVEDUNITELEMENT(#10,-1.);
#10=XTDSIUNIT($,.SECOND.);

/**
* centimetre to the power of four  cm⁴
* Derived dimensional exponents: [4.00 0.00 0.00 0.00 0.00 0.00 0.00]
* XTDDERIVEDUNIT=(Elements)*/
#11=XTDDERIVEDUNIT((#12));
#12=XTDDERIVEDUNITELEMENT(#13,4.);
#13=XTDSIUNIT(.CENTI.,.METRE.);

/**
* kilogram per square metre hour to the power of one half   kg / (m²·√h)
* Derived dimensional exponents: [-2.00 1.00 -0.50 0.00 0.00 0.00 0.00]
* XTDDERIVEDUNIT=(Elements)*/
#14=XTDDERIVEDUNIT((#15,#17,#19));
#15=XTDDERIVEDUNITELEMENT(#16,1.);
#16=XTDSIUNIT(.KILO.,.GRAM.);
#17=XTDDERIVEDUNITELEMENT(#18,-2.);
#18=XTDSIUNIT($,.METRE.);
#19=XTDDERIVEDUNITELEMENT(#20,-0.5);
#20=XTDCONVERSIONBASEDUNIT('hour',3600.,#21);
#21=XTDSIUNIT($,.SECOND.);

/**
* miligram per kilowatt-hour   mg / (kW·h)
* Derived dimensional exponents: [-2.00 0.00 2.00 0.00 0.00 0.00 0.00]
* XTDDERIVEDUNIT=(Elements)*/
#22=XTDDERIVEDUNIT((#23,#25,#27));
#23=XTDDERIVEDUNITELEMENT(#24,1.);
#24=XTDSIUNIT(.MILLI.,.GRAM.);
#25=XTDDERIVEDUNITELEMENT(#26,-1.);
#26=XTDSIUNIT(.KILO.,.WATT.);
#27=XTDDERIVEDUNITELEMENT(#28,-1.);
#28=XTDCONVERSIONBASEDUNIT('hour',3600.,#29);
#29=XTDSIUNIT($,.SECOND.);
```

```
/**
* kilogram per metre   kg/m
* Derived dimensional exponents: [-1.00 1.00 0.00 0.00 0.00 0.00 0.00]
* XTDDERIVEDUNIT=(Elements)*/
#30=XTDDERIVEDUNIT((#31,#33));
#31=XTDDERIVEDUNITELEMENT(#32,1.);
#32=XTDSIUNIT(.KILO.,.GRAM.);
#33=XTDDERIVEDUNITELEMENT(#34,-1.);
#34=XTDSIUNIT($,.METRE.);

/**
* pieces per pack
* Derived dimensional exponents: [0.00 0.00 0.00 0.00 0.00 0.00 0.00]
* Dimensional exponents [0 0 0 0 0 0 0] means dimension = 1 (ISO 80000-1)
* Which is true for dimension-less measures like count
* XTDCONTEXTDEPENDENTUNIT=(Name)*/
#35=XTDCONTEXTDEPENDENTUNIT('PIECES_PER_PACK');

/**
* parts per milion
* Derived dimensional exponents: [0.00 0.00 0.00 0.00 0.00 0.00 0.00]
* XTDCONTEXTDEPENDENTUNIT=(Name)*/
#36=XTDCONTEXTDEPENDENTUNIT('PARTS_PER_MILLION');

/**
* baud
* Derived dimensional exponents: [0.00 0.00 0.00 0.00 0.00 0.00 0.00]
* XTDCONTEXTDEPENDENTUNIT=(Name)*/
#37=XTDCONTEXTDEPENDENTUNIT('BAUD');
ENDSEC;
END-ISO-10303-21;
```