Recommended Practices for User Defined Attributes

Release 1.5

August 15, 2016

Contacts	
Jochen Boy	Phil Rosché
PROSTEP AG	ACCR LLC.
Dolivostraße 11	125 King Charles Circle
64293 Darmstadt / Germany	Summerville, SC 29485 USA
jochen.boy@prostep.com	phil.rosche@accr-llc.com

Table of Contents

List of Figures

Document History

Revision	Date	Change
1.0	2011-10-14	Initial creation
1.1(*)	2012-11-27	Changed section 6.2 to use MLRD instead of SHUO
1.2(*)	2013-02-22	Updated document references
1.3	2014-10-09	Updated Figure 6; editorial changes for publication.
1.4	2015-07-20	Updated section 6.3 for AP242; updates section 8 with additional validation property for measure values.
1.5	2016-08-15	Fixed implementation for integer values; added section 7.4.4; clarifying note to introduction of Annex C.4.1.

(*): Internal review versions; not published.

1 Introduction

This document specifies the recommended practices for the transfer of user defined attributes (UDA's) in Computer Aided Design (CAD) systems. These attributes are usually not derived from the part itself, but are added manually by the CAD system user to supplement the model with additional information. User defined attributes may also serve as a place to store certain information from a native system that has no one-to-one counterpart in the target system.

User defined attributes are usually transferred as key-value pairs, where the key is given by the `name` attribute of the respective `representation_item` that carries the value. The type of attribute is given by the subtype of `representation_item` being used, in particular if it is a measure value (e.g. a `length_measure`). In order to enable a round-trip conversion, it may also be useful to transfer the name of attribute type as defined in the native system.

2 Scope

The following are within scope of this document:

- Transfer of user defined attributes as key-value pairs
- Transfer of meta-data for the attributes, such as the name of the type in the originating system
- Definition of sets of attributes and attribute values
- Assignment of the attribute value to a part, including assembly component instances
- Assignment of the attribute value to a section of the part shape, i.e. solids or surfaces
- Definition of Validation Properties for User Defined Attributes

The following are outside of the scope of this document:

- Transfer of any kind of Validation Properties other than those directly for UDA (for Product Manufacturing Information (PMI) Validation Properties, Assembly Validation Properties and Geometric Validation Properties, see the corresponding Recommended Practices)
- Transfer of Density and Material Identification (see corresponding Recommended Practices)
- Assignment of properties to non-solid or non-surface models
- Assignment of properties to a product, document or feature, or its definition
- Definition of CAD-system specific structures for the association of UDA to geometry

3 Document Identification

For validation purposes, STEP processors shall state which Recommended Practice document and version thereof have been used in the creation of the STEP file. This will not only indicate what information a consumer can expect to find in the file, but even more important where to find it in the file.

This shall be done by adding a pre-defined ID string to the `description` attribute of the `file_description` entity in the STEP file header, which is a list of strings. The ID string consists of four values delimitated by a triple dash ('---'). The values are:

```
Document Type---Document Name---Document Version---Publication Date
```

The string corresponding to this version of this document is:

CAx-IF Rec.Pracs.---User Defined Attributes---1.5---2016-08-15

It will appear in a STEP file as follows:

```
FILE_DESCRIPTION(('...','CAx-IF Rec.Pracs.---User Defined Attributes---1.5---
2014-08-15',),'2;1');
```

4 Fundamental Concepts

The approach used to transfer user defined attributes is the “general property” approach introduced in Part 41. It is based on the concept that an attribute (the key in a key-value pair) is defined once as a placeholder, and is then used to assign the actual values to the respective target elements as often as needed.

The main reference points in a STEP file for which such an attribute may be defined in the given context are:

- the entire part (`product_defintion`)
- an instance of the part in an assembly (`product_definition_relationship`)
- a portion of the shape defining the part (`shape_aspect`)

There are a number of pre-defined property types in STEP that may be used to store a user-defined attribute. In the context of this document, this includes:

- descriptive attributes
  - name and description
- measure values
  - name and value
  - name, value and unit

Note: The `property_definition`, which is the starting point for the definition of an attribute value, has been identified by two means in earlier (pre-0.6) drafts of this document:

- the fact that a `general_property` is associated with it
- the magic string ‘user defined attribute’

This was a redundant definition, since the association of a `general_property` alone already conveys the intent that this is a user defined attribute, and it also considerably limited the potential offered by the use of `general_property`. Hence, the magic string will no longer be used. The new section 5 will explain the details.

5 Definition of the Attribute, Usage and Sets

Even though User Defined Attributes, as the name suggests, are usually defined by the user, they are not entirely arbitrary. There is typically a limited range of attributes, which is then assigned many times to the various elements or instances thereof in the model. These attributes and their values may be extracted by other applications (PDM, downstream processes, etc.) for further use. Sticking with the idea that a UDA is a key-value pair, the approach in STEP is to define the ‘key’ only once and then use it to assign the applicable values to many elements in the model.

However, many CAD systems do not handle user attributes that strictly internally. Two different elements in the model can have two attributes with the same name, but entirely different meanings. To support this concept, the recommendation in the context of the CAx-IF is to keep the 1:1 relationship between `general_property` and `property_definition`. However, the attribute name shall be used as the name attribute values of these entities.

5.1 Definition and Use of General Property

The `general_property` entity will define a 'data field' or 'key' for a user defined attribute. This can then be used to assign a value to one or several elements in the model. The following rules apply to the `general_property.name`:

- it carries the name of the user defined attribute
- it is unique for the (combination of) elements it is assigned to

To assign a value for this attribute:

- create a `property_definition` with the same name as the `general_property` (this is enforced by a where rule in both AP203e2 and AP214)
- link the two together with a `general_property_association` with an empty string as the name

Figure 1 below illustrates the structure needed to define a UDA. The "model element" may be any of the options shown in section 6, and the attribute value and corresponding subtype of `representation_item` may be any of the options shown in section 7 below.

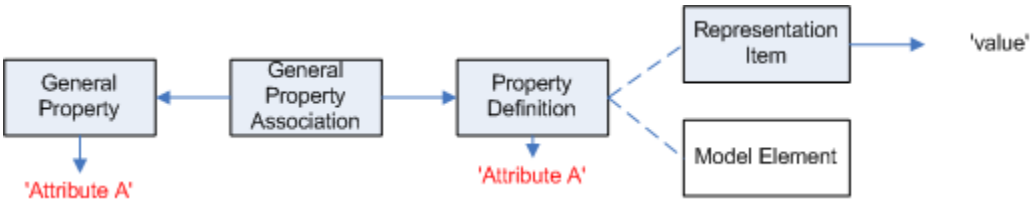

Figure 1: Definition of an attribute name and its usage

Note that PDM systems may follow the original idea in STEP, and define one `general_property` which has many `property_definitions` associated with it. The meaning of this is there is an attribute, which is defined once, and has many values of it assigned to various elements in the model. On import, this should be resolved so that there is one UDA with that name per model element.

5.2 Definition of Attribute Sets

User Defined Attributes can be grouped on two semantic levels: the attribute definition (and thus all its usages) or an individual attribute value.

5.2.1 Groups of Attributes

Note that systems handling user attributes in a way that all attributes with the same name have the same meaning may also define groups on this level. This level of grouping is listed here for completeness. In the context of the CAx-IF, groups of attributes will always be defined on the attribute level, see section 5.2.2 below.

To define a group of attributes in the sense that this grouping shall also be applied to all values of the respective attributes (e.g. the calculated weight, nominal weight and actual weight of a

part), a `general_property` will be created for that group, carrying the name of the group, and relating all `general_property` instances which are members of the group.

The following rules apply to the `general_property_relationship` attribute values:

- `relating_property`: the `general_property` that defines the group of attributes
- `related_property`: the `general_property` that defines an attribute in that group
- `name`: 'decomposition'

5.2.2 Groups of Attribute Values

The approach to define a group of attribute values – i.e. which apply to the specific use of the respective attributes – is quite similar to the grouping of attributes themselves, only it will now happen on the `property_definition` level.

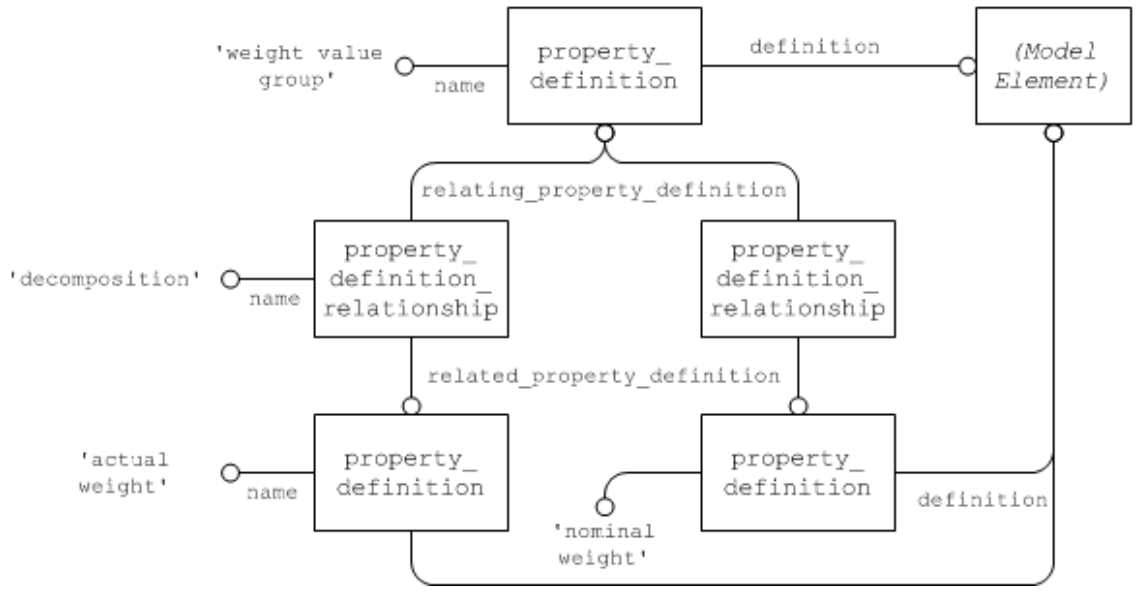

Figure 2: Defining a group of attribute values

Note that all `property_definitons` – the one defining the group and all of the ones defining the attribute values – need to reference the same model element as their `definition` (one of the choices from section 6). This means that only values for the same model element can be grouped.

The following rules apply to the `property_definition_relationship` attributes:

- `relating_property_definition`: the `property_definition` that defines the group of attribute values
- `related_property_definition`: the `property_definition` that defines an attribute value in that group
- `name`: 'decomposition'

6 Specifying the Target for the Attribute

User defined attributes can be attached to the geometry in a STEP file at different levels of granularity, i.e. individual solids or surfaces, or entire parts. While all CAD systems support the definition of attributes at the part level, only some systems can handle attributes at the level of individual shape elements.

6.1 Attributes at the Part/Product Level

The following diagram illustrates the assignment of user defined attributes at the part/product level. The definition of the attribute value, which links to the `property_definition`, is described in section 7.

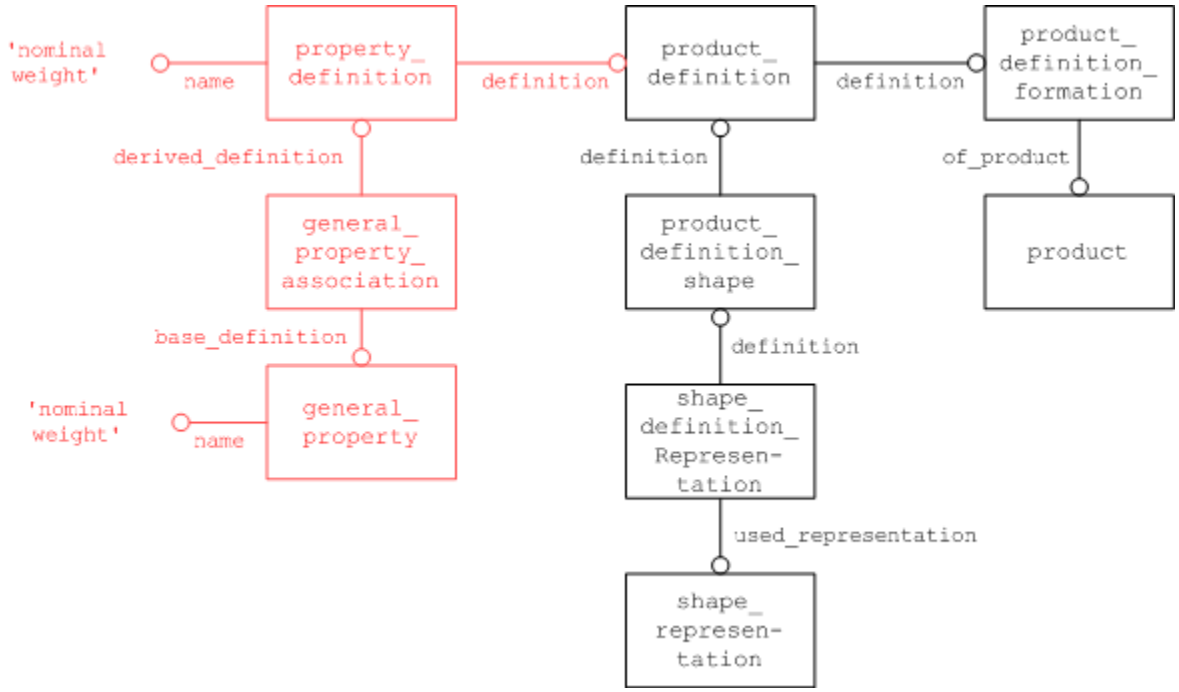

Figure 3: User defined attributes at the part/product level

Part21 Example:

```
#10=PRODUCT('part 1', 'part 1', '', #8);
#20=PRODUCT_DEFINITION_FORMATION('version 1', '', #10);
#30=PRODUCT_DEFINITION('design', $ ,#20, #9);
#40=PRODUCT_DEFINITION_SHAPE('', $, #30);
#50=SHAPE_DEFINITION_REPRESENTATION(#40, #60);
#60=SHAPE_REPRESENTATION('#60, (#895, #442, #447, #452, #889), #891);
#70=PROPERTY_DEFINITION('nominal weight', $,#30);
#80=GENERAL_PROPERTY_ASSOCIATION('', $, #90, #70);
#90=GENERAL_PROPERTY('', 'nominal weight', $);
```

This applies to individual parts as well as assemblies.

6.2 Attributes at Component Instances in an Assembly

If the user defined attribute shall be assigned to a specific instance of a component within an assembly, the property needs to be attached to the assembly definition. If the instance in question is an immediate child of the assembly node, the attribute will be attached to the NAUO:

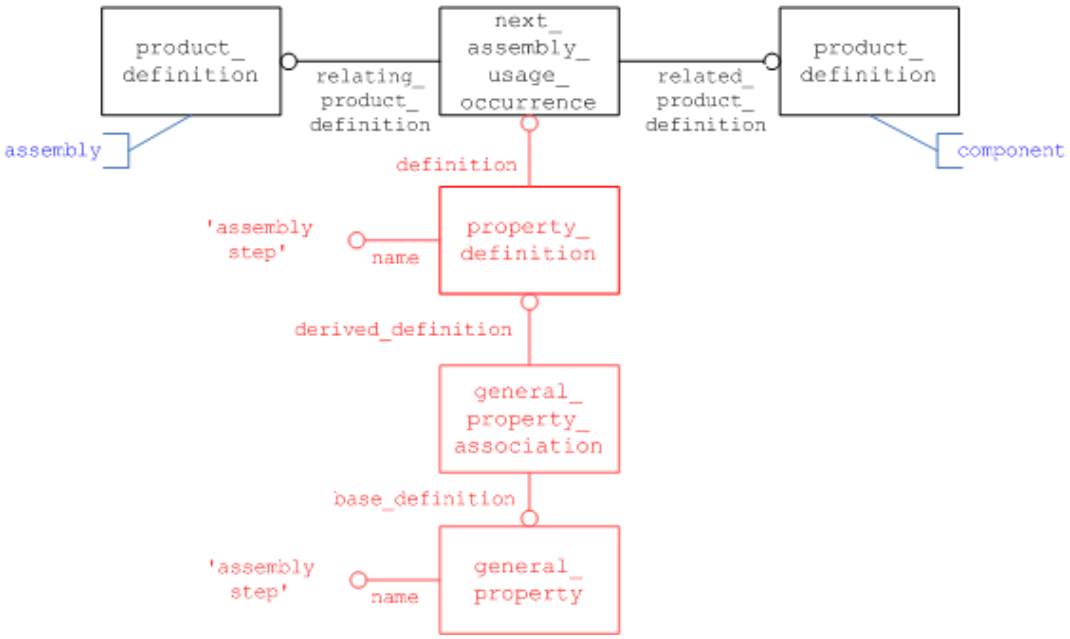

Figure 4: User defined attribute for a simple component instance in an assembly

Part21 Example:

```
#10=PRODUCT('part 1','part 1','',#8);
#20=PRODUCT_DEFINITION_FORMATION('version 1','', #10);
#30=PRODUCT_DEFINITION('design',$,#20,#9);
#40=GENERAL_PROPERTY('','assembly step',$);
#110=PRODUCT('assembly 1','assembly 1','', 8);
#120=PRODUCT_DEFINITION_FORMATION('version 1','',#110);
#130=PRODUCT_DEFINITION('design',$,#120,#9);
#150=NEXT_ASSEMBLY_USAGE_OCCURRENCE('ASS1_PRT1','','Ass1:Prt1',#30,#130,'');
#160=PROPERTY_DEFINITION('assembly step',$,#150);
#170=GENERAL_PROPERTY_ASSOCIATION('',$,#40,#160);
```

If the specific component instance the attribute shall be attached to is several levels down in the assembly tree, the path through the assembly structure from the relative root node to the targeted leaf node needs to be unambiguously identified. This is done by creating an instance of multi_level_reference_designator (MLRD), which references a list of NAUOs. The NAUOs are listed in an ordered manner, from top to bottom.

Note: MLRD was introduced with AP242 DIS. Before that, identification of a component instance deep down in the assembly structure was described using specified_higher_usage_-occurrence (SHUO). In contrast to MLRD with its simple top-down list, SHUO is defined recursively. Though SHUO still is a valid alternative, it was deemed too complex to implement by most CAD vendors. Hence, description of this approach was removed from this document.

The diagram below illustrates the use of MLRD based on the AS1 example:

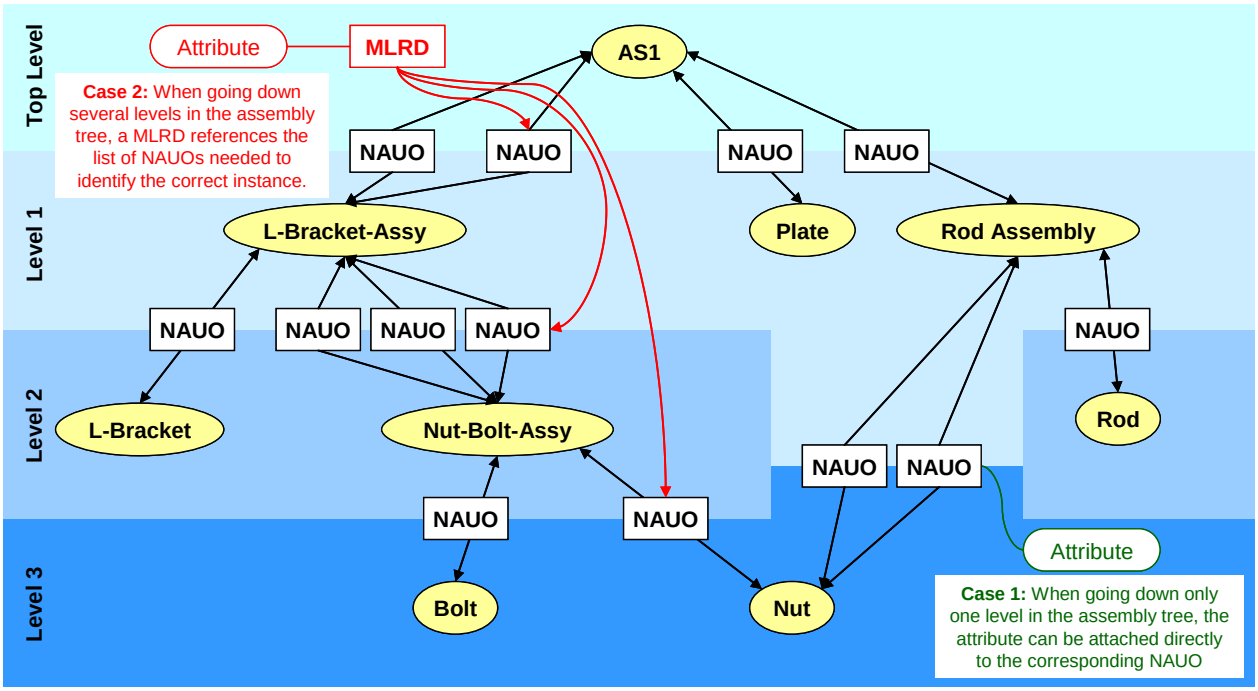

Figure 5: The use of MLRD to identify an instance deeper down in the assembly

MLRD is a subtype of assembly_component_usage, just as NAUO, which means the user defined attribute can be attached to it in the exact same way (see Figure 4). As the name suggests, multi_level_reference_designator requires that for all NAUOs referenced in its location list, the NAUO.reference_designator attribute is populated, and that it is unique in the context of the NAUO.relating_product_definition.

Besides its simpler implementation structure in comparison to the previously used SHUO, MLRD was designed specifically to support the External Element Reference (EER) mechanism. This means that it is possible to define user defined attributes for assembly component instances even in the case where the assembly structure is defined across several files. See Version 3.1 of the Recommended Practices for External (Element) References, in particular section 6.4, for instance identification in nested assemblies.

6.3 Attributes at the Geometry Level

The attachment of user-defined attributes to a single solid, surface or curve within the product geometry is handled via the shape_aspect entity. There are two ways to associate the shape_aspect with its geometric content: the “old” way using a shape_representation, and a newer way using geometric_item_specific_usage.

For details see 6.3.2 and 6.3.3 below.

As stated in the introduction, please note that not every CAD system may be able to find attributes, which are assigned at the solid/surface level.

Note that there are some CAD systems which have additional structuring mechanisms in their model tree beyond the usual part / assembly structure or layers and groups, as they are defined in STEP, and may be capable of assigning user defined attributes to elements of these structures. The so-called “geometrical sets” in CATIA V5 are one example. As there is no equivalent mechanism in STEP for these system-specific structures, and usually no match in other systems

as well, it is recommended to resolve these structures on export to STEP as described above, following the assumption that

Each User Defined Attribute is valid for all Geometric Elements assigned to the `shape_aspect`*.*

Though it is technically possible to define “user practices” which preserve these structures in a CAD to STEP to CAD round-trip exchange, these STEP files would typically not be interoperable with other CAD systems, hence introducing ‘flavors’ to STEP that may lead to unexpected side effects for users not aware of these details. If such practices are employed, they need to be handled with great care and the limitations communicated clearly to all users. It is recommended that an “Implementors Agreement” be established between organizations wishing to exchange this type of data.

Note that in future activities, especially in the context of PMI data exchange, the need to assign UDAs to machining features (hole, thread, pocket, round, fillet, chamfer, etc.) in the model may come up. Support for these types of features is being worked on for AP242 Edition 2. The dedicated entities for the semantic transfer of feature information (e.g. `round_hole`) are subtypes of `shape_aspect`. This means that the structure defined in this section is upward compatible, and exporting the geometry forming such features from a CAD system, as plain `shape_aspects` now is a first step in this direction.

6.3.1 Shape Aspect Identification in AP242

In AP242, there is a uniqueness rule on each of `shape_aspect`, `dimensional_location`, `dimensional_size` and `shape_aspect_relationship`, which requires the attribute pair (`id`, `of_shape`) to be unique if the `id` attribute exists. There is also a global rule requiring uniqueness of the `id` attribute across the population of a collection of the above entity types if the `id` attributes exist. These rules have been introduced in the context of the Semantic Product and Manufacturing Information (PMI) Representation capabilities and External Element References (EER). The second rule is more restrictive as it requires coordination amongst several entity types. For backward compatibility reasons, AP242 does not formally require the `id` attribute to exist.

Since the `id` attribute is derived, an instance of `id_attribute` must be populated, which has the `id` string as its `attribute_value` and any of the aforementioned entity types as `identified_item`.

While adding the `id_attribute` is allowed but not required in the formal AP242 document, omitting it in an AP242 file will violate the business agreement for Semantic PMI and EER. Also, in order not to have to make the decision what purpose a `shape_aspect` is used for, it is recommended to add an `id_attribute` to all instances of the above entities, with an `attribute_value` string that is unique among all instances of `id_attribute` in the context of the respective `product_definition_shape`, i.e. if there are 8 `id_attribute` that reference a combination of the above types which all reference the same `product_definition_shape` in their `of_shape` attribute, there shall be 8 distinct values of `attribute_value`.

There is no business requirement to add `id_attribute` in AP203e2 or AP214 files, since Semantic PMI and EER are out of scope for these APs. It is, however, technically legal to do so.

6.3.2 Geometry Assignment using Shape Representation

This method has been used for Validation Properties at the Geometry Level since their inception. It can associate several geometric elements to a `shape_aspect`, but uses three entities to

do so. Also, a large number of additional `representations` may have an impact on system performance. The structure by which this relationship is shown is given in Figure 6 below.

The geometric elements to be referenced typically are `manifold_solid_brep` for solids and `advanced_face` for surfaces. In general, all types of `geometric_representation_item` are allowed, including edges, shells and surface models.

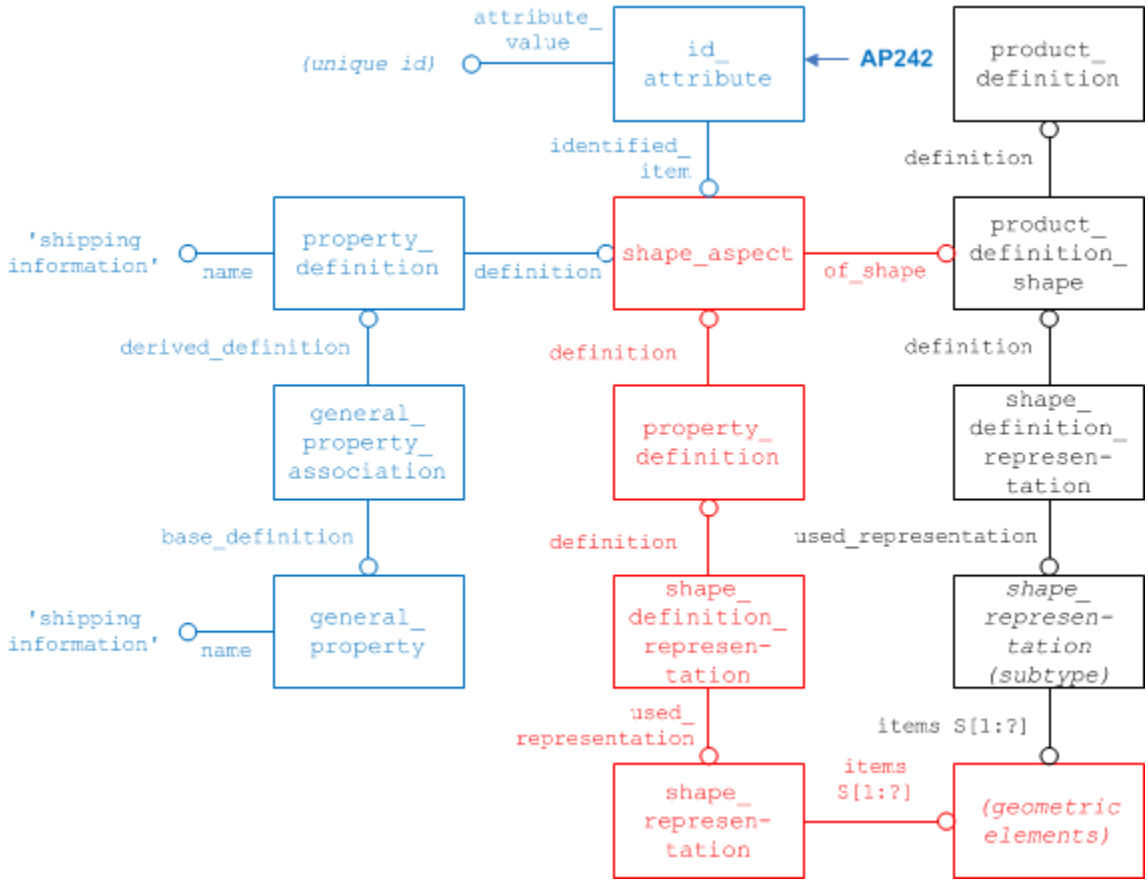

Figure 6: User-Defined Attributes at the Geometry Level (Shape Representation)

Part21 Example:

```
#10=PRODUCT('part 1','part 1','',#8);
#20=PRODUCT_DEFINITION_FORMATION('version 1','',#10);
#30=PRODUCT_DEFINITION('design',$,#20,#9);
#40=GENERAL_PROPERTY('','shipping information',$);
#200=PRODUCT_DEFINITION_SHAPE('',$,#30);
#210=SHAPE_DEFINITION_REPRESENTATION(#200,#220);
#220=ADVANCED_BREP_SHAPE_REPRESENTATION('#220',(#225,#226),#219);
#230=ADVANCED_FACE('#230',(#232), #235, .T.);
#250=SHAPE_ASPECT('face #230',$,#230,.F.);
#251=ID_ATTRIBUTE('sa_ cc503e2531f3',#250);
#255=PROPERTY_DEFINITION('shape for property','',#250);
#260=SHAPE_DEFINITION_REPRESENTATION(#255,#270);
#270=SHAPE_REPRESENTATION('',(#230),#219);
#260=PROPERTY_DEFINITION('shipping information',$,#250);
#290=GENERAL_PROPERTY_ASSOCIATION('',$,#40,#260);
```

Note that in early versions (1.2 and older) of these Recommended Practices, the property_definition between shape_aspect and shape_definition_representation (#255 in the Part21 Example) was accidentally missing. There are some STEP translators who have implemented the structure with the missing property_definiton; this should be supported on import to handle legacy data.

6.3.3 Geometry Assignment using Geometric Item Specific Usage

In AP203 Edition 2 and AP214 Edition 3, the new entity type geometric_item_specific_usage (GISU) was introduced, which was not available in earlier data models. It allows for a much more efficient implementation, as only one entity and no additional representation is needed. Since UDAs are assigned at the geometry level, and are assigned to one specific element (solid, shell, face, curve), GISU can be used with no restrictions.

Note: AP242 introduces a uniqueness rule on GISU, which limits the number of GISU instances per shape_aspect to one. Since a GISU can relate to only a single geometric item, if several geometric elements need to be associated with a shape_aspect, the supertype item_identified_representation_usage has to be used.

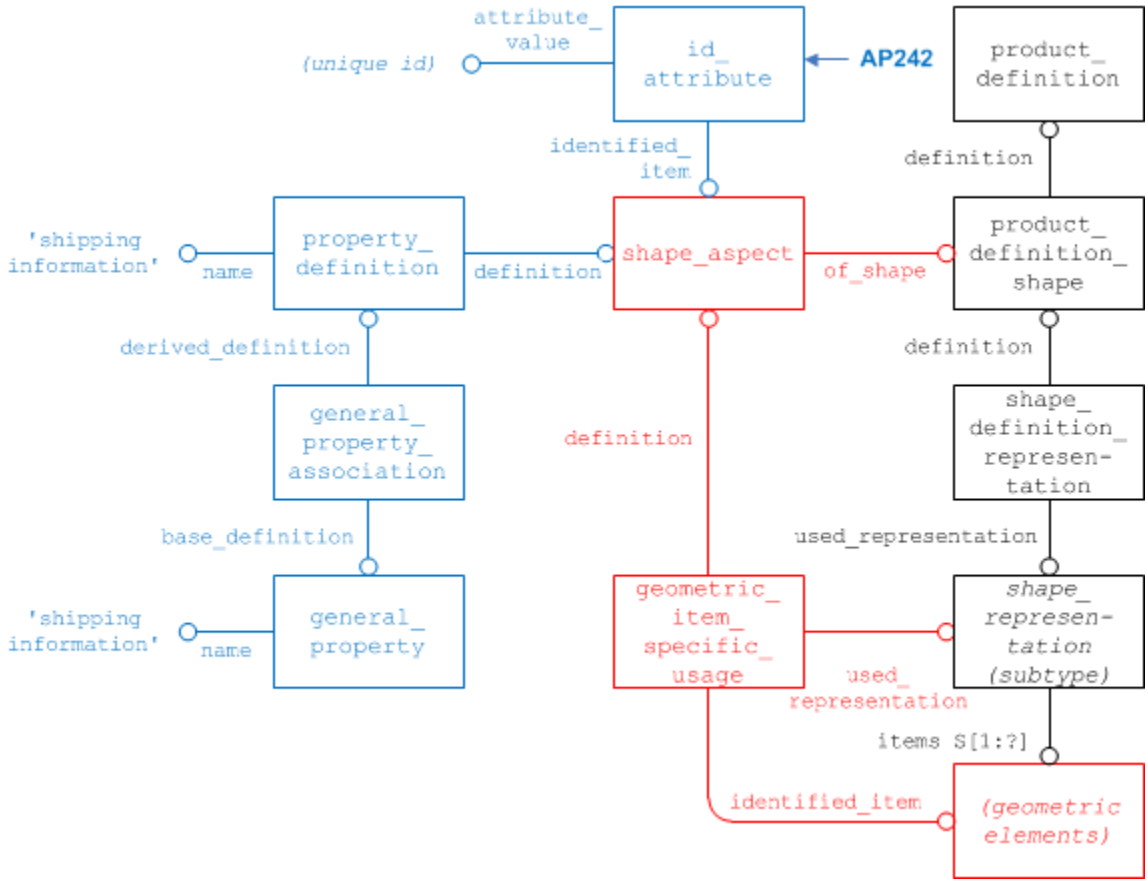

Figure 7: User-Defined Attributes at the Geometry Level (GISU)

Part21 Example:

```
#10=PRODUCT('part 1','part 1','',#8);
#20=PRODUCT_DEFINITION_FORMATION('version 1','',#10);
#30=PRODUCT_DEFINITION('design',$,#20,#9);
#40=GENERAL_PROPERTY('','shipping information',$);
#200=PRODUCT_DEFINITION_SHAPE('',$,#30);
#210=SHAPE_DEFINITION_REPRESENTATION(#200,#220);
#220=ADVANCED_BREP_SHAPE_REPRESENTATION('#220',(#225,#226),#219);
#230=ADVANCED_FACE('#230',(#232), #235, .T.);
#250=SHAPE_ASPECT('face #230',$,#230,.F.);
#251=ID_ATTRIBUTE('sa_cc503e2531f3',#250);
#260=GEOMETRIC_ITEM_SPECIFIC_USAGE('','',#250,#220,#230);
#260=PROPERTY_DEFINITION('shipping information',$,#250);
#290=GENERAL_PROPERTY_ASSOCIATION('',$,#40,#260);
```

7 Definition of the Attribute Value

Depending on the information content of the user defined attribute which shall be transferred, STEP allows for its definition as either name and description, name and value, or name, value and unit. In every case, the anchor entity is the `property_definition` which is highlighted in the figures in section 6 above.

To this `property_definition`, the actual value in the form of the applicable subtype of `representation_item` is then linked through a `property_definition_representation` and a `representation`.

Note that the value for the attribute shall always be defined unambiguously (see section 5.1). These means that usually there will be only one `representation_item` in the set of items of the `representation`. There are, however, cases where an attribute is comprised of several values (e.g. a measure result can have a Boolean value (OK/not OK) and a length measure for the maximum gap). In this case, several `representation_items` may be combined for efficient implementations – as long as it is clear how these values need to be interpreted.

With the currently available STEP schemas, this is indeed rather circumstantial, but unavoidable. The CAx-IF will therefore encourage the standardization groups to pick up the suggestion to invent an entity type `property_definition_with_value`, which will allow streamlining the implementation by attaching the single `representation_item` subtype directly to the `property_definition`. This was in fact proposed before, but rejected at that time.

7.1 Descriptive / String Attribute

A descriptive attribute stores an arbitrary text string in the `description` attribute. As usual in STEP, any special characters in the name or description need to be encoded in Unicode. The name shall be left empty, or, if used, repeat the name of the attribute.

Part21 Example:

```
#10=GENERAL_PROPERTY('','shipping information',$);
#70=PROPERTY_DEFINITION('shipping information',$, #30);
#71=DESCRIPTIVE_REPRESENTATION_ITEM('', 'This Side Up');
#72=REPRESENTATION('', (#71), #162);
#73=PROPERTY_DEFINITION_REPRESENTATION(#70, #72);
#75=GENERAL_PROPERTY_ASSOCIATION('',$, #10, #70);
```

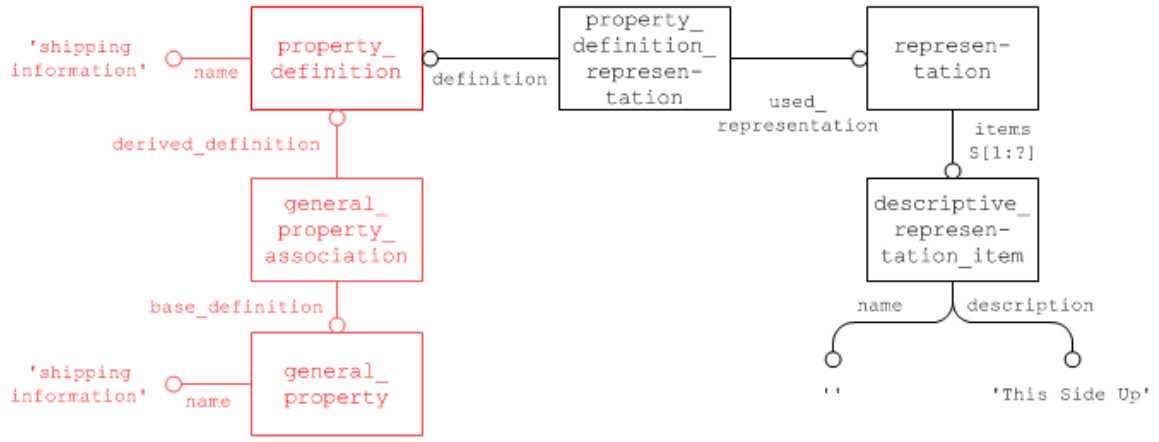

Figure 8: Definition of a user defined descriptive attribute

7.2 Value Attribute

A value attribute transports a general value, which is not a piece of text, i.e. it is either

- an integer value
- a real value, which does not represent a measure value (see 7.3 below for those)
- a Boolean value

AP203 edition 2 and AP242 provide specific subtypes of `representation_item`, that each have a name (again, to be left empty or to repeat the name of the attribute) and an attribute called "`the_value`" which is of the respective type:

- `integer_representation_item.the_value` is of type INTEGER
- `real_representation_item.the_value` is of type REAL
- `boolean_representation_item.the_value` is of type BOOLEAN

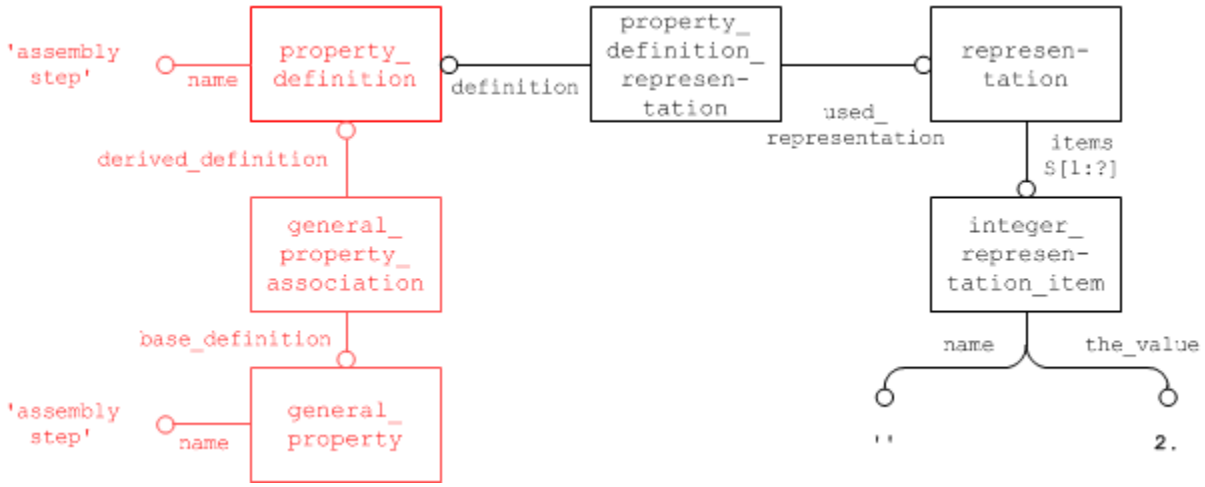

Figure 9: Definition of a user defined value attribute (INTEGER in this case)

Part21 Example:

```
#11=GENERAL_PROPERTY('','assembly step',$);
#160=PROPERTY_DEFINITION('assembly step',$,#150);
#161=INTEGER_REPRESENTATION_ITEM('',2.);
#162=REPRESENTATION('',(#161),#266);
#163=PROPERTY_DEFINITION_REPRESENTATION(#160,#162);
#164=GENERAL_PROPERTY_ASSOCIATION('',$,#11,#160);
```

Note: The value of `integer_representation_item` has to be written as a REAL number, i.e. with trailing decimal point.

- In general, values of type INTEGER are represented in a Part 21 file as integers, i.e. no decimal point. The case of `integer_representation_item`, however, is special. `Integer_representation_item` is a subtype of `int_literal`, which is a subtype of `literal_number`. The latter defines the attribute `the_value` as of type NUMBER; `int_literal` then re-declares that to restrict it to INTEGER. Part 21 defines that in the case of a re-declared attribute, the original (more generic) type shall be used for implementation, here: NUMBER. And NUMBER maps to REAL in Part 21, hence the decimal point. So basically, `integer_representation_item` is an integer with an identity crisis, but from the context (name of the entity) it is clear that the decimal point shall be ignored.
- For compatibility with existing data, `integer_representation_item` values without decimal points shall be supported as well on import.

Note that the specific subtypes listed above are not available in AP214. The following workarounds are suggested for use in AP214 (see also older versions (pre-0.8) of this document):

- for INTEGER, use a `value_representation_item` with `count_measure`. Keep in mind that `count_measure` is of type NUMBER, i.e. it has to be represented as a REAL number in the STEP file. Therefore it is suggested to use an additional attribute (see 7.4 below) to explicitly transfer the information that this shall be interpreted as INTEGER.
- for REAL, use a `value_representation_item` with `numeric_measure`.
- for BOOLEAN, use a `descriptive_representation_item` (as in 7.1), where the value of the `.description` attribute is either “TRUE” or “FALSE”. Again, it is suggested to use an additional attribute (see 7.4 below) to explicitly transfer the information that this shall be interpreted as BOOLEAN.

7.3 Measure Attribute

A measure attribute is given by its name, the measure value, and the measure unit.

Please refer to Annex C of this document for the specific measure types supported in STEP. Please refer to Annex D for the definition of the corresponding units. During the Round25J Review Meeting, the following agreement was made concerning the transfer of values with types:

- If it is possible to define the corresponding measure type and unit in the respective STEP AP, transfer it semantically as described above. **Note** that there is a difference between AP203e2/AP242 and AP214 due to the different versions of Part41 being used.
- If there is no corresponding type / unit, transfer the value including the unit as a text string (see 7.1)

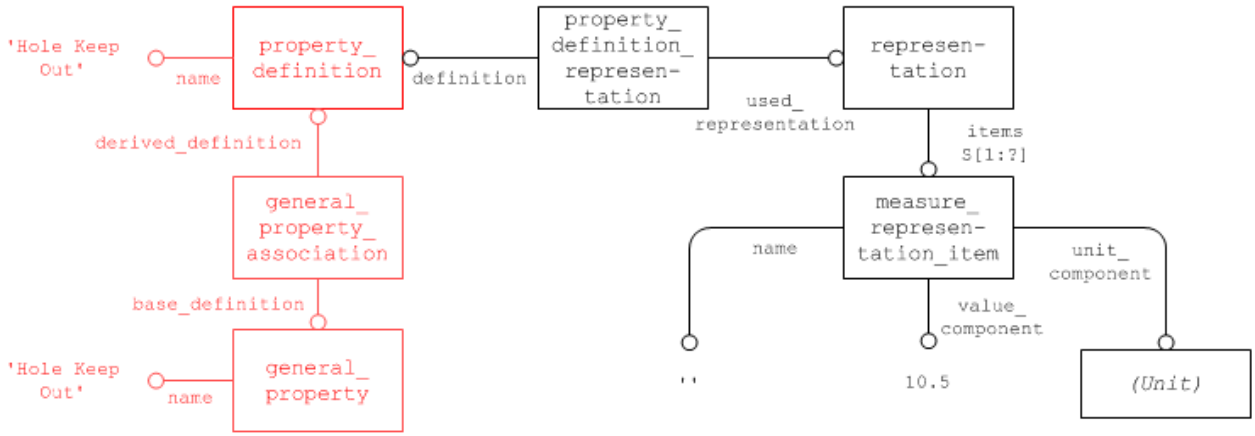

Figure 10: Definition of a user defined measure attribute

Part21 Example:

```
#120=GENERAL_PROPERTY('','Hole Keep Out',$);
#5310=DIMENSIONAL_EXPONENTS(1.E0,0.E0,0.E0,0.E0,0.E0,0.E0,0.E0);
#5320=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.));
#5330=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(2.54E1),#5320);
#5340=(CONVERSION_BASED_UNIT('INCH',#5330)LENGTH_UNIT()NAMED_UNIT(#5310));
#5350=MEASURE_REPRESENTATION_ITEM('',POSITIVE_LENTGH_MEASURE(10.5),#5340);
#5360=REPRESENTATION('',(#5350),#200);
#5370=PROPERTY_DEFINITION('Hole Keep Out',$,#300);
#5380=PROPERTY_DEFINITION_REPRESENTATION(#5370,#5360);
#5390=GENERAL_PROPERTY_ASSOCIATION('',$,#120,#5370);
```

7.4 Transfer of Meta-Data for the User Defined Attributes

In order to enable a round-trip exchange of user defined attributes via STEP in a way that the attributes are mapped onto the same definition as in the native system they originated from, it is also possible to add additional information about the attributes, groups, or their values. This is optional, and follows the same approach as described in 7.1 – but without the use of a `general_property` – and one or more of these may be added to the following elements:

- definition of an attribute (see 5.1): `property_definition` for “meta data” pointing to the `general_property` defining the attribute
- definition of an attribute group (see 5.2.1): `property_definition` for “meta data” pointing to the `general_property` defining the attribute group
- definition of an attribute values group (see 5.2.2): `property_definition` for “meta data” pointing to `property_definition` defining the attribute values group
- an individual attribute value (see 7.1 - 7.3): `property_definition` for “meta data” pointing to `property_definition` defining the attribute value.

7.4.1 Definition

Using this mechanism, it is possible to add even more information about an attribute, an attribute value, or group thereof. This may include CAD-system specific data, such as whether the attribute is relevant for a data management system or not. The identifier of the additional information is carried in the `property_definition.name` attribute, and the value is transferred in the `representation_item` in the set of items of the referenced `representation`.

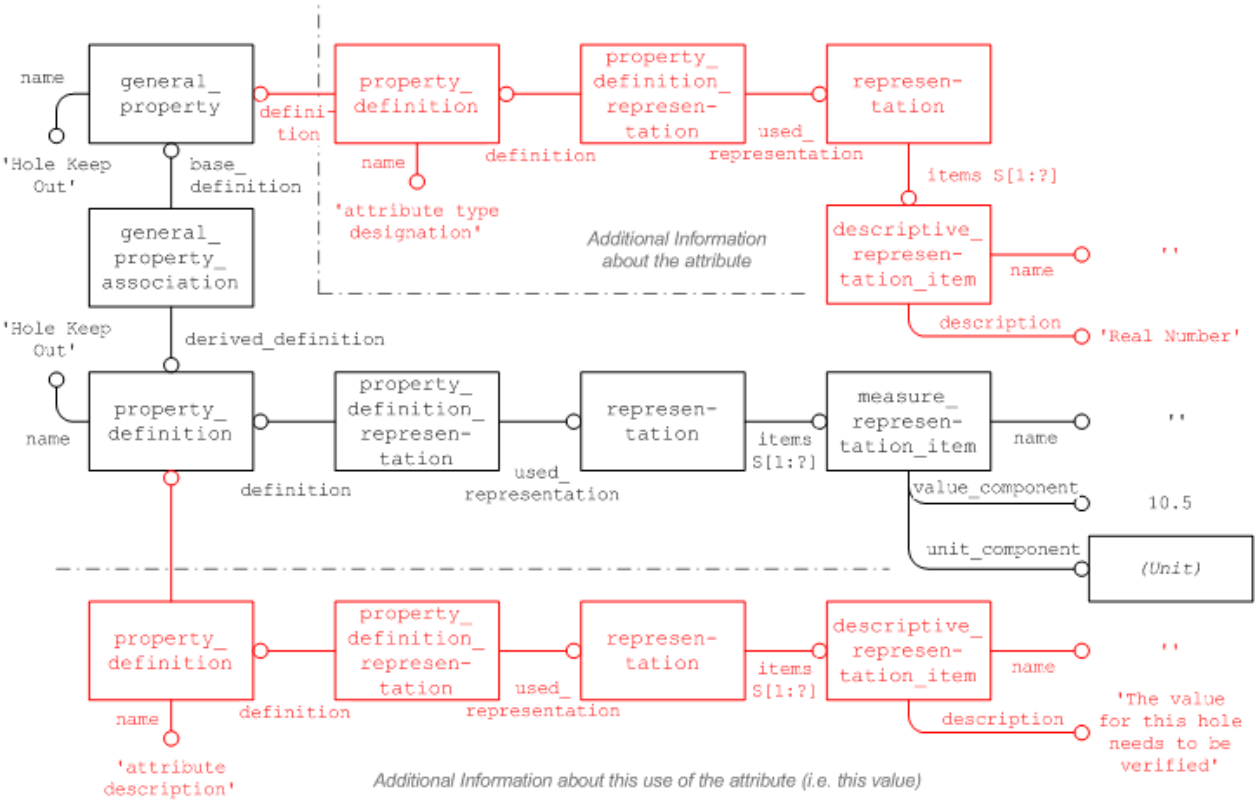

Figure 11: Specification of meta-data for an attribute and one of its values

The “meta data” will be defined as a “property of a property”, and they can be distinguished easily from the actual user defined attributes by two means:

- its `property_defintion` will point to a `general_property` or other `property_definition`, and not one of the model elements identified in section 6.
- its `property_definition` will have no associated `general_property`.

Figure 11 above illustrates this in combination with a measure attribute. Note that with the restrictions in many CAD systems mentioned in section 5, in the CAx-IF meta-data for UDA shall always be attached to the `property_definition` for the attribute value, or – if supported – the group of attribute values.

Note that these “meta-data” attributes will not be taken into account for the UDA Validation Properties (see section 8). The following sections will give two common examples:

7.4.2 Designation of the Attribute Type

In order to transfer the name of the type for the user defined attribute as given in the originating system, add an additional property including a `descriptive_representation_item` with the following attribute characteristics:

- property_definition.name: ‘attribute type designation’
- descriptive_representation_item.description: The designation of the attribute type as given in the native system

This shall be linked to the attribute value definition (property_defintion).

7.4.3 Attribute / Value / Group Description

In order to transfer a description for the attribute (**Note** that this is a description *about* the attribute, in contrast to a descriptive attribute as defined in section 7.1), add an additional property containing a descriptive_representation_item with the following attribute values:

- property_definition.name: ‘attribute description’
- descriptive_representation_item.description: Textual information about the attribute

Again, in the context of the current CAx-IF scope, this shall be linked to the property_definition of the UDA or a group of values.

7.4.4 Specifying Number of Decimal Places

For certain attributes – measure values or REAL values – it might be required to specify the number of decimal places using a value type qualifier. This is done using a complex entity for the respective representation_item subtype (measure, real or value), containing the qualified_representation_item entity, which in turn, points to a value_format_type_qualifier. This specifies the number of places to the right and left of the decimal point by its format_type attribute.

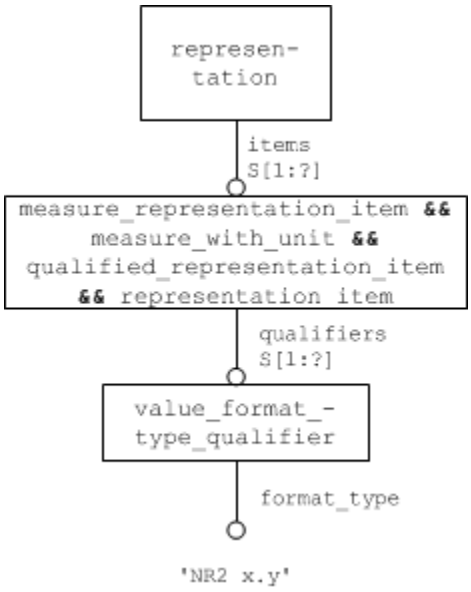

Figure 12: Definition of a value format type qualifier

The values for x and y in Figure 12 specify the number of places to the left and right of the decimal point respectively. A typical value would be ‘NR2 2.2’.

8 UDA Validation Properties

Since User Defined Attributes (UDA) may be used to transfer significant information, which is relevant for either downstream applications or long-term archiving purposes, it shall be ensured that no properties are lost during transfer. The STEP file structure for this count measure is analogous to similar validation properties, e.g. the “number of children” assembly validation property.

All UDA Validation Properties shall be attached at the part / product level so that all systems will be capable of finding them.

There shall be two main groups of UDA Validation Properties:

[1] a count of UDAs for each model element type, i.e. how many UDAs are assigned to parts, component instances, solids, faces, curves, etc.

[2] a count of UDAs for each main class of attribute data types (string, integer, real, boolean).

[3] a separate count of UDAs that are measure attributes per section 7.3, in order to distinguish numerical values with and without units.

The following rules apply to the counting:

- the number of attribute values (as defined in sections 7.1-7.3) shall be counted in total. The `general_properties` and “meta data” (as in 7.4) will not be taken into account.
- the two sums of the two counts (“Element Sum” = sum of all UDAs per model element counts [1] and “Type Sum” = sum of all UDAs per attribute data type counts [2]) have to match.
    - The separate count of measure attributes [3] will not be taken into account for the matching of the sums, as these attributes are actually counted twice.

Note: Since all UDA Validation Properties are counts, they will be represented as `integer_representation_items` in the file. Refer to the notes in section 7.2 concerning the correct implementation of `integer_representation_item`, or for alternative implementation in AP214.

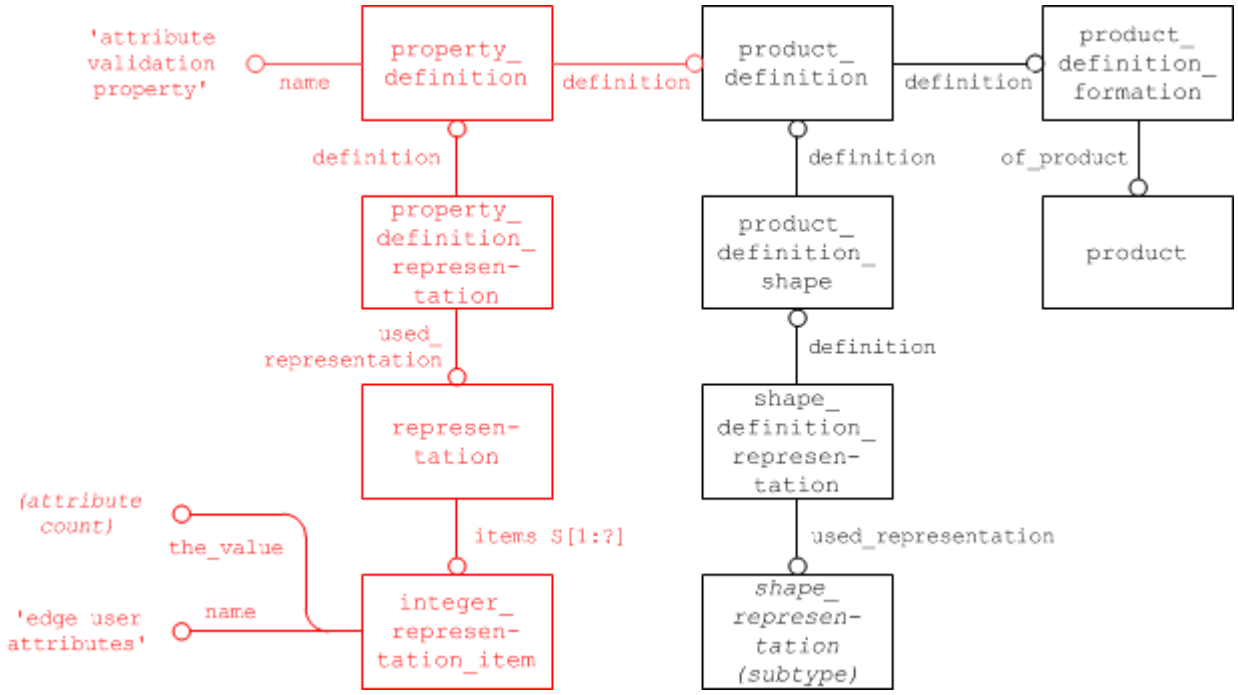

Figure 13: Definition of user defined Attribute Validation Properties

The name of the `property_definition` shall be “attribute validation property”. The name of the `integer_representation_item` depends on what is being counted:

UDA Model Element Types

value_representation_item.name	Counts UDA at …
‘vertex user attributes’	vertices and / or points (elements of dim. 0)
‘edge user attributes’	edges and / or curves (elements of dim. 1)
‘face user attributes’	faces and / or surfaces (elements of dim. 2)
‘solid user attributes’	solids and / or shells (elements of dim. 3)
‘part user attributes’	the part/product level.
‘instance user attributes’	the instance of components in the assembly in context of the product they are counted at.

Note that, as with Geometric Validation Properties, some CAD systems are not capable of handling properties assigned to individual geometric elements, but only at the part/product level. These systems should either disregard the geometry-level UDA validation properties, or flag the resulting “errors” as a system limitation in the log file.

Note that UDAs at the assembly instance level shall be counted at the product (assembly node) which defines the context of their use. For example, looking at Figure 5, assume there is one UDA at the NAUO from AS1 to an L-Bracket Assembly, and another UDA at the MLRD which connects the AS1 root node to a specific usage of the Nut. The “instance user attributes” count at the product representing the AS1 root node would then be “2”.

If a system misses UDAs attached to MLRD due to the way it handles these properties internally, the resulting “error” from the validation properties should be clearly marked as a system limitation in the log file.

A similar count is being proposed when groups of attributes (see 5.2.1) or groups of attribute values (see 5.2.2) are used:

value_representation_item.name	Counts …
‘user attribute groups’	…how many groups of attribute values are defined for this part
‘group user attributes’	…how many attributes are in the group. Note that this validation property (see Figure 13) has to be linked to the `property_definition` that defines the group.

Note that systems not handling groups of UDAs should either disregard these values, or again flag the resulting “errors” as a system limitation in the log file.

UDA Data Type Classes

value_representation_item.name	Counts UDA at that contain…
‘integer user attributes’	integer values (see 7.2)
‘real user attributes’	real numbers (see 7.2 and 7.3)
‘text user attributes’	arbitrary text (see 7.1)
‘boolean user attributes’	a boolean value (see 7.2)
‘measure value user attributes’	a measure value (see 7.3)

Note the count for ‘real user attributes’ shall include all numeric values that are represented by a real number, regardless of whether they have a unit attached (see 7.3) or not (see 7.2).

Version 1.4 of this document introduced the additional count for ‘measure value user attributes’, which will count only real numbers with an assigned unit (see 7.3). Since those are now counted twice, the measure value count shall not be included in the “type sum” for matching with the “element sum” as described above.

9 Usage of UDA in combination with External References

In many integration scenarios, STEP is well-established as a process-accompanying neutral format to exchange metadata across domains and organizations. The user defined attributes as defined in this document are a subset of this important information.

This section summarizes how to transfer part-level UDA via STEP when the assembly data is stored in one or several STEP files with external references and the part files are in native format. This is a typical scenario in industries where large assemblies often occur.

In order to ensure consistency, it is important that the part-level UDA are always specified in the referenced file, so that they are defined only once for all occurrences of that part (see section 6.2 for UDA that apply only to certain instances of a component).

This enables the following scenarios:

- Nested external references, where the assembly structure is split into many files, and it is thus likely that there will be several files referencing the same part. With the definition of the UDA on the referenced side, they need to be defined only once.
- Whenever the process needs to access the UDA without the need (or the capability) to open the native data
- Updating the UDA without having to update the assembly data

9.1 Intermediate File Approach

The chosen approach uses an “intermediate file”, as shown in Figure 14 below:

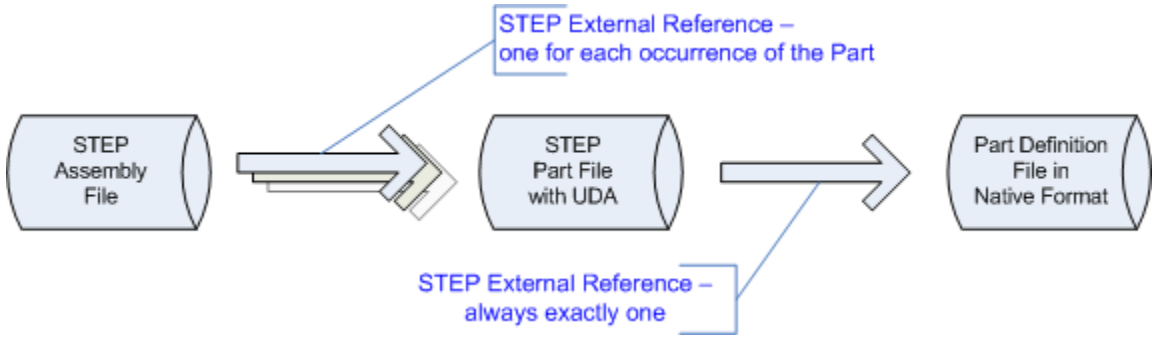

Figure 14: Intermediate File for External References with Part-Level UDA

The entity structure in the referencing STEP assembly file remains unchanged from the usual external references (compare to Figure 3 in the Rec. Practices for External References, v2.1), except that it will point to the STEP file instead of the native file representing the part.

The intermediate STEP file with the UDA will be quite small and contain only the following information:

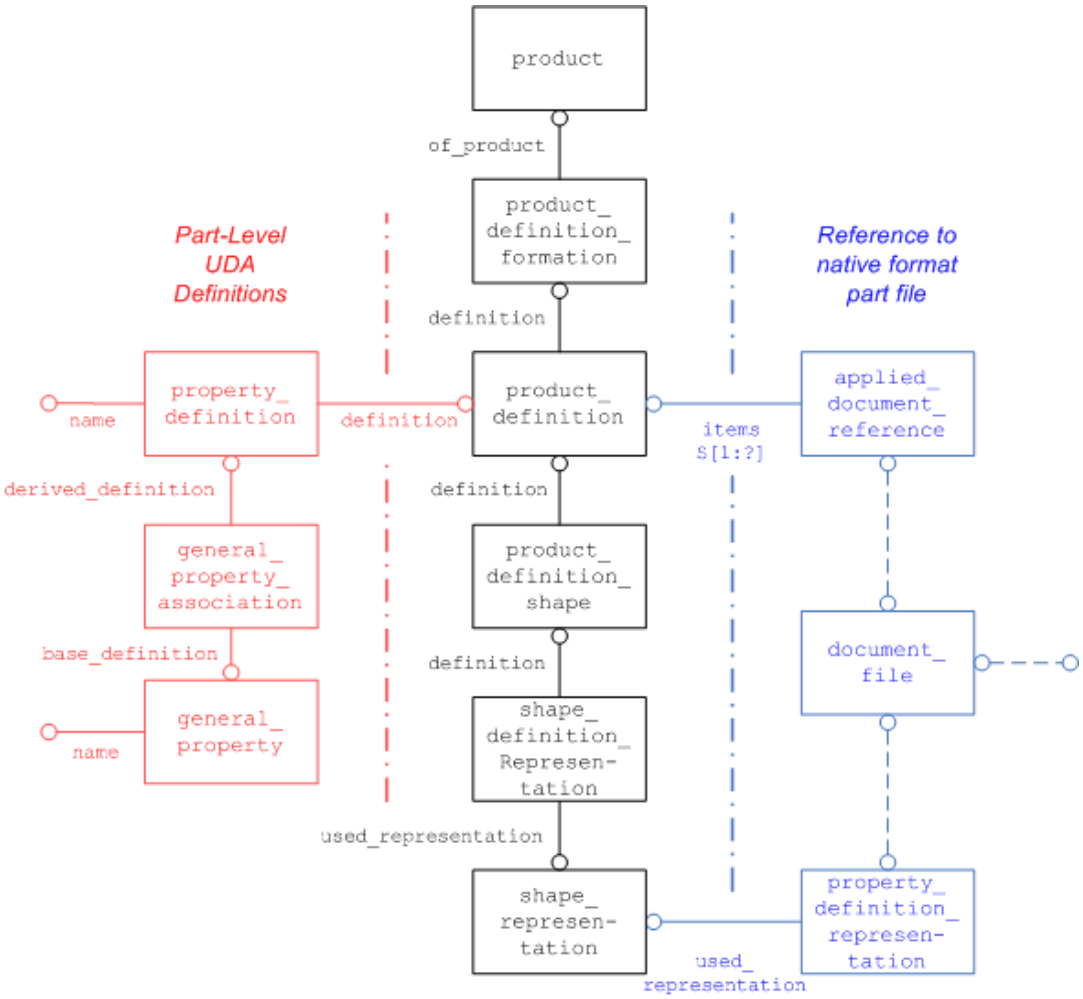

Figure 15: Structure of the Intermediate File

9.2 Known Limitations

The “intermediate file” approach described above will work for part-level UDA with external references (entire assembly structure in one file) and nested external references (assembly structure broken down into individual files per assembly level).

For UDA at assembly component instance level, there is a clear limitation at the moment as this will work with external references only in the case where the entire assembly structure is given in one file, so that the SHUO approach can be used (cp. Figure 5 in section 6.2). In the case of nested external references, this is not possible as there is currently no way to define the correct instance path through the assembly structure across several files.

Should such an approach become available in the future, it will be added to this document.

Annex A Part 21 File Examples

STEP files relating to the capabilities described in this document are available in the public STEP File Library on the CAx-IF homepage; see either

- http://www.cax-if.de/library/ or
- http://www.cax-if.org/library/

The files are typically based on AP203 Edition 2, AP214 Edition3, or AP242, and will have been checked for syntax and compliance with the Recommended Practices.

Annex B Availability of implementation schemas

B.1 AP214

The AP214 schemas support the implementation of the capabilities as described. The schemas can be retrieved from:

- IS Version (2001) – http://www.cax-if.de/documents/ap214_is_schema.zip
- 3rd Edition (2010) – http://www.cax-if.de/documents/AP214E3_2010.zip

B.2 AP203 2nd Edition

The long form EXPRESS schema for the second edition of AP203 can be retrieved from:

- http://www.cax-if.de/documents/part403ts_wg3n2635mim_lf.exp

Note that the first edition of AP203 is no longer support in the Recommended Practices.

B.3 AP242

The long form EXPRESS schema for the first edition of AP242 can be retrieved from:

- http://www.cax-if.de/documents/ap242_is_mim_lf_v1.36.zip

Annex C Measure Value Types available in AP214 and AP203e2

The following types for a measure value are defined in section 21.3 of Part 41.

Note that AP214 only supports a subset of these types. These are underlined in the list below.

Entity Type	Definition
absorbed_dose_measure	An absorbed_dose_measure is the value of the absorbed dose of radiation
acceleration_measure	An acceleration_measure is the value of the rate of change of velocity
amount_of_substance_-measure	An amount_of_substance_measure is the value for the quantity of a substance when compared with the number of atoms in 0.012 kilogram of carbon 12
area_measure	An area_measure is the value of the extent of a surface
capacitance_measure	A capacitance_measure is the value of capacitance
celsius_temperature_-measure	A celsius_temperature_measure is the value for the degree of heat of a body
conductance_measure	A conductance_measure is the value of an electrical conductance
context_dependent_measure	A context_dependent_measure is the value of a physical quantity that may be interpreted based on the context in which it is used
count_measure	A count_measure is the value of a count
descriptive_measure	A descriptive_measure is a textual value of a physical quantity
dose_equivalent_measure	A dose_equivalent_measure is the value of the radiation dose equivalent
electric_charge_measure	An electric_charge_measure is the value of an electrical charge
electric_current_measure	An electric_current_measure is the value for the movement of electrically charged particles
electric_potential_measure	An electric_potential_measure is the value of an electrical potential
energy_measure	An energy_measure is the value of energy, or work done, in a system
force_measure	A force_measure is the value of a force
frequency_measure	A frequency_measure is the value of a frequency
illuminance_measure	An illuminance_measure is the value of illuminance
inductance_measure	An inductance_measure is the value of inductance
length_measure	A length_measure is the value of a distance
luminous_flux_measure	A luminous_flux_measure is the value of luminous flux
luminous_intensity_measure	A luminous_intensity_measure is the value for the brightness of a body
magnetic_flux_density_-measure	A magnetic_flux_density_measure is the value of magnetic flux density
magnetic_flux_measure	A magnetic_flux_measure is the value of magnetic flux

Entity Type	Definition
mass_measure	A mass_measure is the value of the amount of matter that a body contains
non_negative_length_-measure	A non_negative_length_measure type is a length_measure whose value is greater than or equal to zero
numeric_measure	A numeric_measure is the numeric value of a physical quantity
parameter_value	A parameter_value is the value which specifies the amount of a parameter in a parameter space
plane_angle_measure	A plane_angle_measure is the value of an angle in a plane
positive_length_measure	A positive_length_measure is a length_measure that is greater than zero
positive_plane_angle_-measure	A positive_plane_angle_measure is a plane_angle_measure that is greater than zero
positive_ratio_measure	A positive_ratio_measure is a ratio_measure that is greater than zero
power_measure	A power_measure is the value of power, or the rate of doing work
pressure_measure	A pressure_measure is the value of force per unit area
radioactivity_measure	A radioactivity_measure is the value of the radioactive disintegration
ratio_measure	A ratio_measure is the value of the relation between two physical quantities that are of the same kind
resistance_measure	A resistance_measure is the value of electrical resistance
solid_angle_measure	A solid_angle_measure is the value of a solid angle
thermodynamic_temperature_-measure	A thermodynamic_temperature_measure is the value for the degree of heat of a body
time_measure	A time_measure id the value of the duration of periods
velocity_measure	A velocity_measure is the value of the rate of change of position
volume_measure	A volume_measure is the value of the solid content of a body

Annex D Recommendation for the Definition of Units[1]

This clause provides recommendations for instance population for the definition of units in the data set. Once the definition is created, other data instances reference the units as required.

Note: The definitions given in this Annex are valid for the following schema versions:

- AP214 3rd Edition (2010)
- AP203 2nd Edition (later than Nov. 2008)
- AP242 (all versions)

The definitions hereafter do **not** apply to AP214 IS (2001) and any version of AP203 before end of 2008, as they use an older version of Part 41.

Definitions for area and volume units for AP214 IS (2001) are given in **Annex D.4.3.**

D.1 SI Base Unit Definitions

The following is the recommendation for exchange of SI base unit definitions:

Base unit reference instances:

```
#4  =(LENGTH_UNIT()                NAMED_UNIT(*)          SI_UNIT($,.METRE.));
#14 =(MASS_UNIT()                  NAMED_UNIT(*)     SI_UNIT(.KILO.,.GRAM.)); [2]
#24 =(NAMED_UNIT(*)                SI_UNIT($, .SECOND.)          TIME_UNIT());
#26 =(ELECTRIC_CURRENT_UNIT()      NAMED_UNIT(*)       SI_UNIT($, .AMPERE.));
#426=(NAMED_UNIT(*) SI_UNIT($, .KELVIN.) THERMODYNAMIC_TEMPERATURE_UNIT());
#427=(AMOUNT_OF_SUBSTANCE_UNIT() NAMED_UNIT(*)          SI_UNIT($, .MOLE.));
#428=(LUMINOUS_INTENSITY_UNIT()  NAMED_UNIT(*)      SI_UNIT($, .CANDELA.));
```

D.2 SI Derived Units

SI derived unit exchange should use the `derived_unit` and `unit_elements` referencing either a SI base unit or other SI derived units rather than relying directly on `dimensional_exponents`[3].

D.2.1 SI Derived Unit using User Defined Names

The list of entries in `si_unit_name` in Part 41 is not exhaustive. In the case that the name of a derived unit is not included in `si_unit_name` then an instance of `derived_unit` (that is not also an instance of `si_unit`) shall be populated. In that case, the `derived_unit.name` attribute shall be populated to identify the unit.

---

[1] http://physics.nist.gov/cuu/Units/units.html

[2] This instance is created to support definition of SI derived units and is the formal definition that the kilogram is the SI unit of mass.

[3] TC1 for Part41 ed3 addresses the kilogram issue by isolating the application of .KILO. used to define a unit from the application of .KILO. used as a prefix in a measure value. It also corrected invalid data structures for `area_unit` and `volume_unit`.

D.2.2 SI Derived Unit using Predefined Names

In the case that the name of a derived unit is included in `si_unit_name`, then an instance of the name specific subtype of `derived_unit` and `si_unit` shall be populated. In that case, the `derived_unit.name` is set equal to the `si_unit_name` by the schema and any population of `derived_unit.name` is ignored. The recommendation is to not populate the `derived_unit.name` attribute.

List of SI derived units whose names are included in `si_unit_name`:

- absorbed_dose_unit
- radioactivity_unit
- capacitance_unit
- dose_equivalent_unit
- electric_charge_unit
- conductance_unit
- electric_potential_unit
- energy_unit
- magnetic_flux_density_unit
- force_unit
- frequency_unit
- illuminance_unit
- inductance_unit
- magnetic_flux_unit
- power_unit
- pressure_unit
- resistance_unit

D.3 Derived Units whose System of Units is Unspecified

The following derived units are included in Part 41 but their system of units is unspecified in Part 41[4]:

- acceleration
- area
- velocity
- volume

D.3.1 Receiver expected to infer SI Derived Units for Unspecified Units

If all `derived_unit_element` instances reference SI units, then the derived unit is an SI derived unit.

D.3.2 Receiver expected to infer some Non SI Units for Unspecified Units

For the case that other units are exchanged (e.g., English engineering) each of the `derived_unit_element` instances referenced by the `derived_unit` should be in the same system of units.

[4] Part 41 specifies the fact that it is a `derived_unit` and the `dimensional_exponents` values for the unit.

D.4 Detailed Examples of Measure Unit Definitions

D.4.1 Definition of SI Units

Definition of “Newton”:

A Newton is [kg*m*sec^{-2}]. Part 41 provides the ability to explicitly state that force is derived, that the Newton is a SI derived unit with a name.

Each instance of `si_unit` that is referenced by a `derived_unit_element` supporting a derive SI unit shall not provide a prefix unless the `si_unit` being referenced is also a `mass_unit`, in which case the prefix is required to be `.KILO.`

Part 41 *requires* to populate kilogram as the mass unit when Newton is defined so as to make the mathematical properties of the data set consistent with SI system of units[5].

Recommended approach:

```
/* establish system of units */
#5=DERIVED_UNIT_ELEMENT(#4,1.0);
#15=DERIVED_UNIT_ELEMENT(#14,1.0);
#25=DERIVED_UNIT_ELEMENT(#24,-2.0);
/* establish newton as SI force unit */
#4161100=SI_FORCE_UNIT((#5,#15,#25),*,$,.NEWTON.);
```

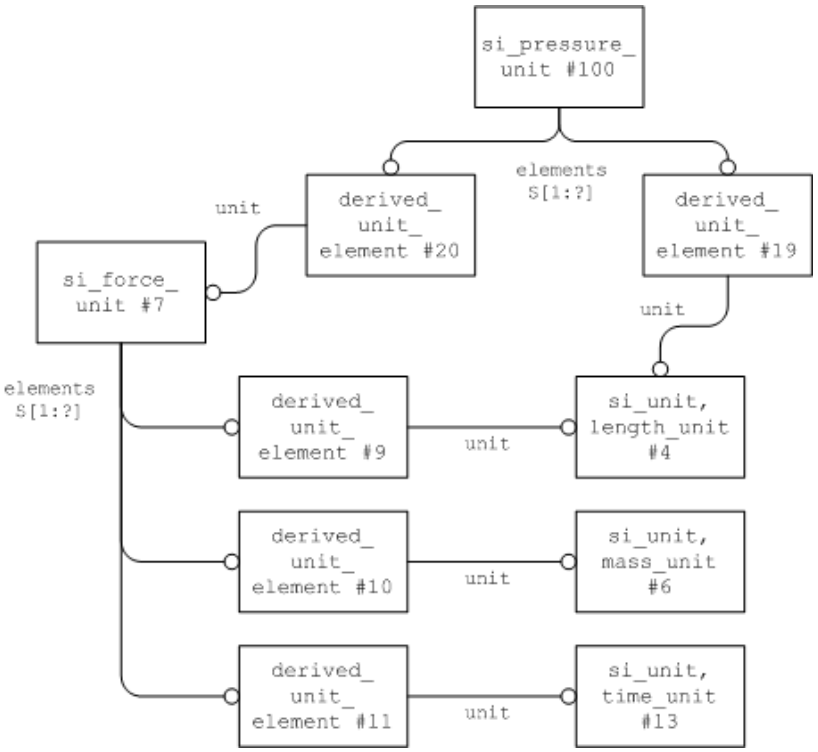

Figure 16: SI Unit Definition for "Pascal"

[5] If the mass_unit prefix is not provided, even though the Newton is declared to be the unit, the numerical instance data declares the unit to be the dyne.

Definition of “Pascal”:

A Pascal is 1 N/m^2. Therefore a Pascal is [kg*m^{-1}*sec^{-2}]. The recommended approach is to derive a Pascal from a Newton previously defined.

```
/* establish the division by m^2. */
#550005=DERIVED_UNIT_ELEMENT(#4,-2.0);
/* establish a reference to a Newton already defined */
#5500025=DERIVED_UNIT_ELEMENT(#4161100,1.0);
#4161200=SI_PRESSURE_UNIT((#550005,#5500025),*,$,.PASCAL.);
```

Definition of “Joule”:

A Joule is 1 N*m. Therefore a Joule is [kg*m^2*sec^{-2}]. The Joule may be derived from SI base units or may be derived from a Newton previously defined. The recommended approach is to derive a Joule from a Newton previously defined.

```
/* establish the multiplication by m. */
#650005=DERIVED_UNIT_ELEMENT(#4,1.0);
/* establish a reference to a newton already defined */
#6500025=DERIVED_UNIT_ELEMENT(#4161100,1.0);
#4161300=SI_ENERGY_UNIT((#650005,#6500025),*,$,.JOULE.);
```

Definition of “Watt”:

A Watt is 1 Joule/sec. Therefore a Watt is [kg*m^2*sec^{-3}]. The Watt may be derived from SI base units or may be derived from a Joule previously defined. The recommended approach is to derive a Watt from a Joule previously defined.

```
/* establish the division by sec. */
#750005=DERIVED_UNIT_ELEMENT(#24,-1.0);
/* establish a reference to a joule already defined */
#7500025=DERIVED_UNIT_ELEMENT(#4161300,1.0);
#4161400=SI_POWER_UNIT((#750005,#7500025),*,$,.WATT.);
```

Definition of “Coulomb”:

A Coulomb is 1 amp*sec. The Coulomb is derived from SI base units. The coulomb itself does not require population of kilogram but is included herein because it may be used in the derivation of capacitance.

```
/* establish system of units */
#8500015=DERIVED_UNIT_ELEMENT(#24,1.0);
#8500025=DERIVED_UNIT_ELEMENT(#26,1.0);
/* establish coulomb as SI electric charge unit */
#1001=SI_ELECTRIC_CHARGE_UNIT((#8500015,#8500025),*,$,.COULOMB.);
```

Definition of “Volt”:

A Volt is 1 Watt/Amp. Therefore a Volt is [kg*m^2*sec^{-3}*amp^{-1}]. The Volt may be derived from SI base units or may be derived from a Watt previously defined. The recommended approach is to derive a Volt from a Watt previously defined.

```
/* establish the division by amp. */
#950005=DERIVED_UNIT_ELEMENT(#26,-1.0);
/* establish a reference to a watt already defined */
#9500025=DERIVED_UNIT_ELEMENT(#4161400,1.0);
#1002=SI_ELECTRIC_POTENTIAL_UNIT((#950005,#9500025),*,$,.VOLT.);
```

Definition of “Farad”:

A Farad is 1 Coulomb/Volt. Therefore a Farad is [$kg^{-1}*m^{-2}*sec^{4}*amp^{2}$]. The Farad may be derived from SI base units or may be derived from a Coulomb and Volt previously defined. The recommended approach is to derive a Farad from a Coulomb and Volt previously defined.

```
/* establish a reference to a coulomb already defined. */
#860005=DERIVED_UNIT_ELEMENT(#1001,1.0);
/* establish a reference to a volt already defined */
#8600025=DERIVED_UNIT_ELEMENT(#1002,-1.0);
#4161500=SI_CAPACITANCE_UNIT((#860005,#8600025),*,$,.FARAD.);
```

Definition of “Ohm”:

An Ohm is 1 Volt/Amp. Therefore an Ohm is $kg*m^{2}*sec^{-3}*amp^{-2}$]. The Ohm may be derived from SI base units or may be derived from a Volt previously defined. The recommended approach is to derive an Ohm from a Volt previously defined.

```
/* establish a reference to a volt already defined. */
#870005=DERIVED_UNIT_ELEMENT(#1002,1.0);
/* establish a reference to amp*/
#8700025=DERIVED_UNIT_ELEMENT(#26,-1.0);
#10099=SI_RESISTANCE_UNIT((#870005,#8700025),*,$,.OHM.);
```

Definition of “Siemens”:

A Siemens is 1 Amp/Volt. Therefore a Siemens is [$kg^{-1}*m^{-2}*sec^{3}*amp^{2}$]. The Siemens may be derived from SI base units or may be derived from a Volt previously defined, or may be derived from an Ohm previously defined. The recommended approach is to derive a Siemens from an Ohm previously defined.

```
/* establish a reference to an ohm already defined. */
#880005=DERIVED_UNIT_ELEMENT(#10099,-1.0);
#100=SI_CONDUCTANCE_UNIT((#880005),*,$,.SIEMENS.);
```

Definition of “Weber”:

A Weber is 1 Volt*Second. Therefore a Weber is [$kg*m^{2}*sec^{-2}*amp^{-1}$]. The Weber may be derived from SI base units or may be derived from a Volt previously defined. The recommended approach is to derive a Weber from a Volt previously defined.

```
/* establish a reference to a volt already defined. */
#890005=DERIVED_UNIT_ELEMENT(#1002,1.0);
/* establish a reference to second*/
#8900025=DERIVED_UNIT_ELEMENT(#24,1.0);
#10023=SI_MAGNETIC_FLUX_UNIT((#890005,#8900025),*,$,.WEBER.);
```

Definition of “Tesla”:

A Tesla is 1 Weber/Meter^{2}. Therefore a Tesla is [$kg*sec^{-2}*amp^{-1}$]. The Tesla may be derived from SI base units or may be derived from a Weber previously defined. The recommended approach is to derive a Tesla from a Weber previously defined.

```
/* establish a reference to a weber already defined. */
#900005=DERIVED_UNIT_ELEMENT(#10023,1.0);
/* establish a reference to metre*/
#9000025=DERIVED_UNIT_ELEMENT(#4,-2.0);
#4161600=SI_MAGNETIC_FLUX_DENSITY_UNIT((#900005,#9000025),*,$,.TESLA.);
```

Definition of “Henry”:

A Henry is 1 Weber/Amp. Therefore a Henry is [kg*m^{2}*sec^{-2}*amp^{-2}]. The Henry may be derived from SI base units or may be derived from a Weber previously defined. The recommended approach is to derive a Henry from a Weber previously defined.

```
/* establish a reference to a weber already defined. */
#910005=DERIVED_UNIT_ELEMENT(#10023,1.0);
/* establish a reference to ampere*/
#9100025=DERIVED_UNIT_ELEMENT(#26,-1.0);
#47000=SI_INDUCTANCE_UNIT((#910005,#9100025),*,$,.HENRY.);
```

D.4.2 Non SI unit definitions

Well-known are the definition of length, area and volume. Examples included are square millimeters, cubic millimeters, inches, square inches and cubic inches.

Definition of Square Millimeter:

```
#613=DERIVED_UNIT_ELEMENT(#4,2.0);
#614=AREA_UNIT((#613));
#615=NAME_ATTRIBUTE('SQUARE MILLIMETRE',#614);
```

Note In this example #614 defines an area unit of square millimeter. `Area_unit` is an instance of `derived_unit` but is not an instance of `si_unit` so `derived_unit.name` is populated.

Definition of Cubic Millimeter:

```
#610=DERIVED_UNIT_ELEMENT(#4,3.0);
#611=VOLUME_UNIT((#610));
#612=NAME_ATTRIBUTE('CUBIC MILLIMETRE',#611);
```

Note In this example #614 defines a volume unit of cubic millimeter. `Volume_unit` is an instance of `derived_unit` but is not an instance of `si_unit` so `derived_unit.name` is populated.

Definition of Inch:

```
#71 =DIMENSIONAL_EXPONENTS(1.,0.,0.,0.,0.,0.,0.) ;--length
#2944=(LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI.,.METRE.));
/* Because the unit is an si_unit, the dimensional exponents for #2944 are not
exchanged but are calculated based on the enumeration values. */
#2945=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(2.54E1),#2944);
#2946=(CONVERSION_BASED_UNIT('INCH',#2945) LENGTH_UNIT() NAMED_UNIT(#71));
```

Note There is a (new) local rule on `conversion_based_unit` requiring the `dimensional_exponents` of a `conversion_based_unit` to be equal to the `dimensional_exponents` of the `unit_component` attribute of the `conversion_factor`. This example satisfies that rule because record #2946 references #71 directly and `dimensional_exponents` are derived by the receiver (if need be) for record #2944.

Definition of Square Inch:

```
#6130=DERIVED_UNIT_ELEMENT(#2946,2.0);
#6140=AREA_UNIT((#6130));
#6150=NAME_ATTRIBUTE('SQUARE INCH',#6140);
```

Note In this example #6140 defines an area unit of square inch. `Area_unit` is an instance of `derived_unit` but is not an instance of `si_unit` so `derived_unit.name` is populated.

Definition of Cubic Inch:

```
#6100=DERIVED_UNIT_ELEMENT(#2946,3.0);
#6110=VOLUME_UNIT((#6100));
#6120=NAME_ATTRIBUTE('CUBIC INCH',#6110);
```

Note In this example #6110 actually defines a volume unit of cubic inch. `Volume_unit` is an instance of `derived_unit` but is not an instance of `si_unit` so `derived_unit.name` is populated.

Definition of "Pound Force":

Recommended approach using existing Newton declaration (#100):

```
/* conversion from newton to 'pound force' where 'pounds force is defined per
the English Gravitational System.  */
#111=DIMENSIONAL_EXPONENTS(1.,1.,-2.,0.,0.,0.,0.) ;--force
#101=FORCE_MEASURE_WITH_UNIT(FORCE_MEASURE(4.4482216152605),#100);
#103=(CONVERSION_BASED_UNIT('pound force',#101) FORCE_UNIT()
NAMED_UNIT(#111));
```

D.4.3 Non SI unit definitions for AP214 IS (2001)

The IS version of AP214, published in 2001, uses an older version of Part 41. This has an impact on the instantiation of units since some entity types have been changed. For instance, `area_unit` is a subtype of `named_unit` in AP214-IS, while in all APs using the new Part 41, it is a subtype of `derived_unit`.

The instantiations of area and volume units for AP214-IS are given below. Those are the most widely used units, as they are needed for Geometric Validation Properties. Instantiation examples for other types of measure can be given upon request.

Definition of Square Millimeter:

```
#130=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.));
#200=DERIVED_UNIT((#210));
#210=DERIVED_UNIT_ELEMENT(#130,2.);
```

Note In this example #200 defines an area unit of square millimeter.

Definition of Cubic Millimeter:

```
#130=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.));
#220=DERIVED_UNIT((#230));
#230=DERIVED_UNIT_ELEMENT(#130,3.);
```

Note In this example #220 defines an area unit of square millimeter.

Definition of Inch:

```
#709=(CONVERSION_BASED_UNIT('INCH',#712) LENGTH_UNIT ()NAMED_UNIT(#710));
#710=DIMENSIONAL_EXPONENTS(1.,0.,0.,0.,0.,0.);
#712=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(25.4),#713);
#713=(LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI.,.METRE.));
```

Definition of Square Inch:

```
#209=DERIVED_UNIT_ELEMENT(#219,2.);
#211=NAME_ATTRIBUTE('SQUARE INCH',#213);
#213=DERIVED_UNIT((#209));
#219=(CONVERSION_BASED_UNIT('INCH',#226) LENGTH_UNIT() NAMED_UNIT(#223));
#223=DIMENSIONAL_EXPONENTS(1.,0.,0.,0.,0.,0.);
#226=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(25.4),#229);
#229=(LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI.,.METRE.));
```

Note In this example #213 defines an area unit of square inch.

Definition of Cubic Inch:

```
#208=DERIVED_UNIT_ELEMENT(#47218,3.);
#210=NAME_ATTRIBUTE('CUBIC INCH',#212);
#212=DERIVED_UNIT((#208));
#218=(CONVERSION_BASED_UNIT('INCH',#225) LENGTH_UNIT() NAMED_UNIT(#222));
#222=DIMENSIONAL_EXPONENTS(1.,0.,0.,0.,0.,0.);
#225=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(25.4),#228);
#228=(LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI.,.METRE.));
```

Note In this example #212 actually defines a volume unit of cubic inch.

D.5 Example Application of Unit Definitions to Measure Value

```
#714=AREA_MEASURE_WITH_UNIT(AREA_MEASURE(150.0),#614);
```

In this example #714 describes the surface area of a small cube with 5mm sides

```
#711=VOLUME_MEASURE_WITH_UNIT(VOLUME_MEASURE(125.0),#611);
```

In this example #711 describes the volume of a small cube with 5mm sides

```
#7140=AREA_MEASURE_WITH_UNIT(AREA_MEASURE(150.0),#6140);
```

In this example #7140 describes the surface area of a small cube with 5in sides

```
#7110=VOLUME_MEASURE_WITH_UNIT(VOLUME_MEASURE(125.0),#6110);
```

In this example #7110 describes the volume of a small cube with 5in sides.

```
#8000=FORCE_MEASURE_WITH_UNIT(FORCE_MEASURE(250.2),#103);
```

In this example #8000 describes the force of 250.2 pounds force produced by an engine.

D.6 Measure schema errata

The current (Amp) exponent for Farad has an error in the `FUNCTION dimensions_for_si_unit` and in the `FUNCTION valid_units`. The correct value is 2, whilst the AP203e2 and AP214 schemas have 1. This error was corrected with ISO 10303-41ed3TC2. It is recommended to patch the longform schema manually.