Recommended Practices
for
Model Styling
And Organization

Release 1.9

8 March 2021

CAx-IF

Jochen Boy
PROSTEP AG
jochen.boy@prostep.com

Robert Lipman
NIST
robert.lipman@nist.gov

Phil Rosché
ACCR, LLC.
phil.rosche@accr-llc.com

Table of Contents

List of Figures

List of Tables

Document History

This document replaces the following CAx-IF Recommended Practices:

- Recommended Practices for Colors and Layers;
  published September 24, 2001
- Recommended Practices for Assembly Instance Styling; Release 1.0;
  published November 19, 2002
- Recommended Practices for Colors, Layers and Groups; Draft Release 1.1;
  published December 1, 2008

The current document covers the scope of the preceding ones and adds new and updated concepts.

Revision	Date	Change
1.0	2011-05-23	Initial creation
1.1	2011-09-05	Update of Invisibility Definition
1.2	2011-12-15	Addition of Point Style & Point Color
1.3	2012-11-01	Replacement of Assembly Instance Styling Section
1.4	2014-01-23	Addition of Transparency
1.5	2016-08-15	Moved Overriding Styles section; added Styling for Axis Placements; added clarification to Layers section; added CATIA Geometric Sets.
1.6	2018-08-17	Fixed instantiation for null_styles and styling of axis placements (Figures 1, 5, 9, and 10).
1.7	2020-03-12	Added notes for clarification to 4.1 and 4.2.4; added reference to AP242 Edition 2.
1.8	2020-05-29	Added note on ‘global’ styling containers in assemblies; added pre- and postprocessor recommendations to 4.2.4; added definition of Geometry for Geometric Sets (section 8).
1.9	2021-03-08	Refined definitions for “null” styles and use of invisibility (section 4.1, 4.2.1 and 5.2.1) Changed definition of Geometry for Geometric Sets (section 8).

1 Introduction

This document describes the recommended practices for implementing the ability to exchange information about model styling – colors and visibility – and model organization – layers and groups – via the STEP standard.

The support of Colors and Layers is standard functionality in most STEP processors for many years. These capabilities support a better presentation of the model on the screen, and an organization of the elements in the model as per the user companies' guidelines. Groups represent additional organizational structures within the model supported by some CAD systems.

This document also describes the recommended practices for implementing the ability to assign context dependent styles to individual instances of a component in an assembly via the STEP standard.

The approaches described hereafter have been combined from previously separate, but related, documents and updated to the latest agreements and changes in the data structure.

2 Scope

The following are within the scope of this document:

- Specification of colors for solids and topologically bounded surfaces including their constituent shape elements as well as geometrically bounded wireframe and surface data
- The definition of transparency and reflection characteristics for surfaces
- The definition of point styles and colors
- The assignment of styles (colors, visibility) to instances of a component in an assembly
- The definition of layers and groups

The following are out of scope for this document:

- The definition of "Saved Views" (as defined in digital product definition standards) for model viewing organization. These are described in the "Recommended Practices for the Representation and Presentation of Product and Manufacturing Information (PMI)".
- The definition of surface texture.

3 Document Identification

For validation purposes, STEP processors shall state which Recommended Practice document and version 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.---Model Styling and Organization---1.9---2021-03-08

It will appear in a STEP file as follows:

```
FILE_DESCRIPTION(('...','CAx-IF Rec.Pracs.---Model Styling and
Organization---1.9---2021-03-08',),'2;1');
```

4 Model Styling

The following sections cover the aspects of displaying the model described in the STEP file in the intended way. This is typically done by assigning colors at the top level, which are then inherited down the model structure to the individual geometric elements. For individual elements, this styling can be overridden in general, or in a specific context. The applicable styles also include invisibility.

4.1 Global Styling Container

It is a general convention in STEP – to be precise in Part46 – that only styled items shall be displayed when viewing the contents of a STEP file. That means that all geometric elements that do not have any style assigned at all shall be invisible.

If a style is assigned, there are two options: either, an explicit style is assigned – for instance a specific color – defining the way the model shall be shown, or a "null" style is used as a kind of placeholder, meaning that it is up to the receiving system to determine the way the data is displayed if no other explicit style is assigned . As this concept is quite important, it is summarized in the following table:

Assigned Style	Model or model element is displayed….
none	not at all
"null" style	according to target system preferences (i.e. default system colors) if no explicit styles are found in the subsequent model structure (also see section 4.2.1)
explicit style (color…)	as defined in the STEP file

Table 1: Styling choices

In every STEP file, there shall be at least one global styling container, meaning an entity that displays all information about the initial display of the model. This container can be one of two entity types:

- `draughting_model` ("DM")
- `mechanical_design_geometric_presentation_representation` ("MDGPR")

As there may be more than one DM or MDGPR in a STEP file, it is important to clearly identify the one acting as the global styling container. This is achieved in the following way:

- The `name` attribute is an empty string (")
- If there are other instances of DM or MDGPR in the file, the global one is always referenced by the `rep_2` attribute of `mechanical_design_and_draughting_relationship` (or its supertype, `representation_relationship`).
- Any occurrences of `draughting_model_item_association` will refer to the global DM.

Note that any additional items in the set of items of the DM or MDGPR besides the part geometry (e.g. annotations, camera models…) as well as any advanced implementation approaches including `presentation_set` and `presentation_area` are out of scope for this document. Please refer to the "Recommended Practices for the Representation and Presentation of Product and Manufacturing Information (PMI)", section 10.4 ("Saved Views") for further details.

Figure 1 below illustrates the minimum set of display information required in a STEP file. It defines that the entire part shape shall be displayed, by linking the top-level `shape_representation` (typically `advanced_brep_shape_representation` ("ABSR") for solid

models) to the global styling container. Definition of the `null_style` means that the target system's preferred or default settings will be applied to display the data.

The structure in Figure 1 shall be created for all top-level representations containing geometry. This also applies to supplemental geometry (`constructive_geometry_representation`, cp. corresponding Recommended Practices), as well as surface or wireframe models. For an assembly, it is sufficient to include the top node `shape_representation`. By convention this will then apply to all child nodes as well.

Important Note The styling container and the shape representations mapped into it need to share the same `geometric_representation_context`. The `mapped_item.mapping_target` and `representation_map.mapping_origin` therefore shall define a unit transformation, i.e. with (0/0/0) as origin and the unit vectors as directions.

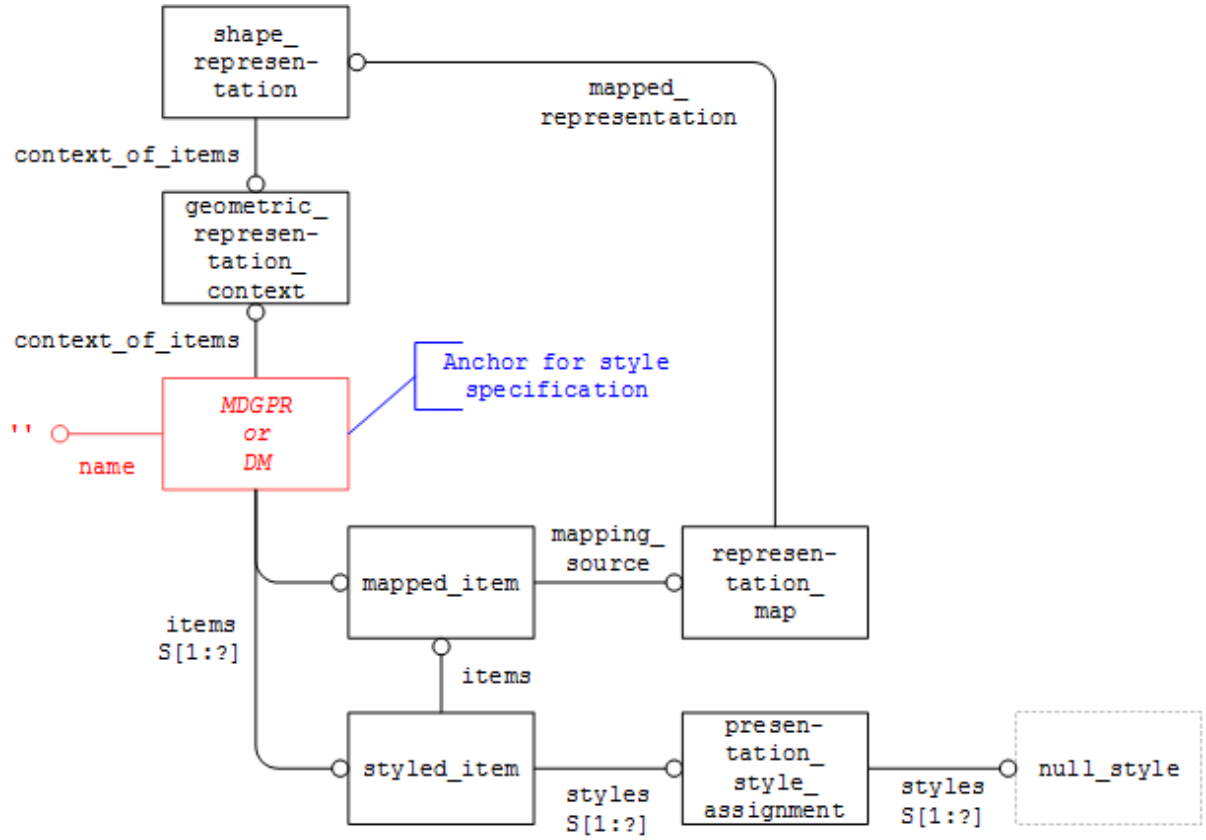

Figure 1: Global Styling Container Definition

Note that in the case of assemblies, each component part / assembly node will have its own 'global' styling container, sharing the respective `geometric_representation_context`. Styles defined at the assembly level can override styles defined within a part (see section 5), but the part styling definition shall always be self-contained.

4.2 Coloring of Solids, Surfaces and Curves

4.2.1 Priority and inheritance of model colors

The styling information is always inherited from top to bottom, i.e. along the assembly structure, and within component parts, the solid or surface styles are propagated down to the constituent faces and edges. The priority list for styling elements is as follows:

1. Assembly node (`shape_representation`)
2. Component part (`shape_representation`)
3. Solids (`manifold_solid_brep`, `brep_with_voids`) OR surfaces (`shell_based_surface_model`)
4. Surfaces only (`open_shell`, `closed_shell`)
5. Faces / edges
6. Geometric surfaces / curves
7. Points / vertices

Solids and surfaces may be colored by using `fill_area_style_colour`, where faces lying on a solid should be styled by overriding the solid style, as are the edges. The edges are treated as being independent of the face due to them potentially being used by two faces. This could lead to a conflict when the two faces were colored differently. See sections 4.2.4 and 4.3.2 for handling of overriding styles.

In order to maintain similarity of style, this approach also applies to geometrically bounded surfaces and wireframes. The priority list for coloring these elements is as follows:

1. Wireframe / Surface collector (`geometric_set`, `geometric_curve_set`)
2. Geometric Representation Item (e.g. `trimmed_curve`, `b_spline_surface`)

Colors should be instantiated from the top down for these hierarchies, e.g. the solid should always be colored, then any differences in face colors applied by overriding the solid color for the different face. Similarly, for wireframe, the `geometric_set` / `geometric_curve_set` should always be colored in the majority color for the geometric entities, those deviating from this majority color being overridden.

The top-down idea applies to assemblies. If an assembly node is styled with a specific color, then all its constituents will be displayed with that color, regardless of how they were styled at part-level. Assigning a “null” style at the assembly level will maintain the appearance defined within each constituent. See chapter 5 on how to style individual component instances in an assembly.

4.2.2 Color Instantiation

Colors can be defined for the surfaces of a model by assigning it to the respective solid, shell or surface, based on the priority list given in section 4.2.1 above.

Note that the two sides of a surface can have different colors assigned; the default recommendation is to assign the same color to both sides (`surface_side = .BOTH.`)

Figure 2 below illustrates the structure to assign a specific color to a surface:

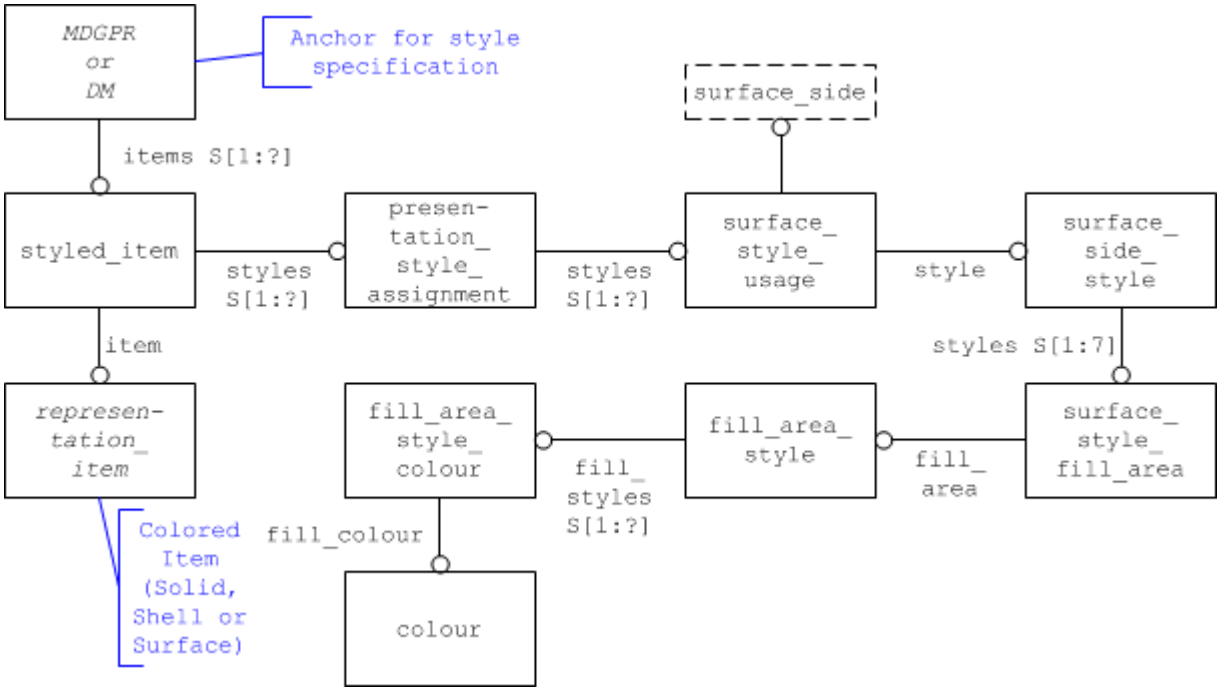

Figure 2: Color Representation for Surfaces / Solids

Note: Faces of a solid or shell usually inherit their color from the superordinate model element as stated in 4.2.1. If a particular face of a solid or shell shall have a different color, overriding face color as defined in section 4.3 below has to be applied.

Part21 Example:

```
#164=MANIFOLD_SOLID_BREP('',#163);
#226=DRAUGHTING_PRE_DEFINED_COLOUR('cyan');
#227=FILL_AREA_STYLE_COLOUR('',#226);
#228=FILL_AREA_STYLE('',(#227));
#229=SURFACE_STYLE_FILL_AREA(#228);
#230=SURFACE_SIDE_STYLE('',(#229));
#231=SURFACE_STYLE_USAGE(.BOTH.,#230);
#232=PRESENTATION_STYLE_ASSIGNMENT((#231));
#233=STYLED_ITEM('',(#232),#164);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268,#273,),#26
9);
```

Edges, curves and sets of curves in a solid, surface or wireframe model can be assigned a color by using the `curve_style` entity.

Note: Edge curves usually inherit their color from the face they are the edge of (see 4.2.1). If the edge shall have a different color than its face, overriding edge color as defined in section 4.3.2 below has to be applied.

Figure 3 below illustrates the structure to assign a specific color to a surface:

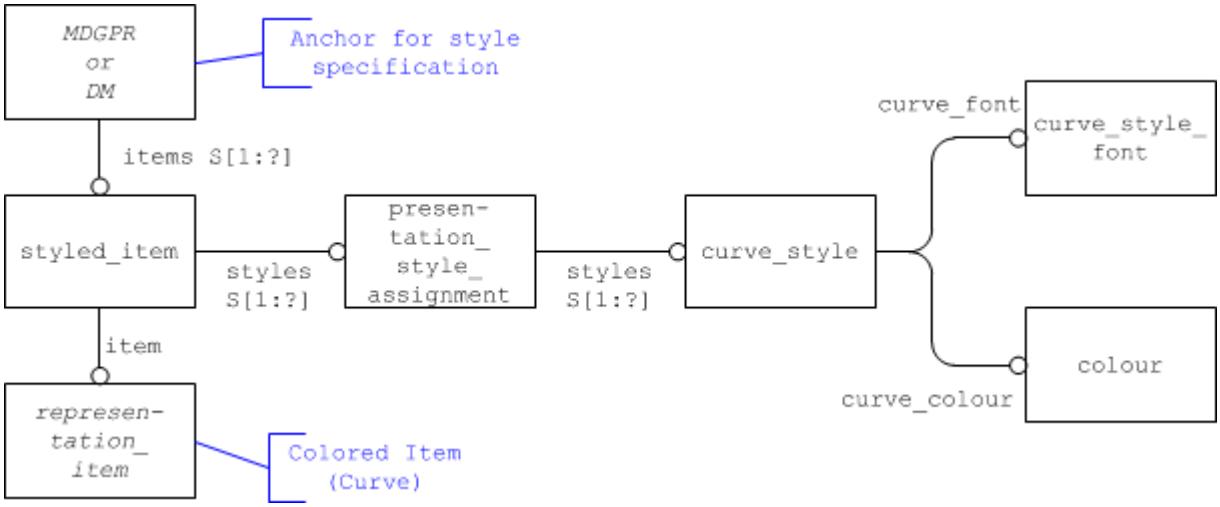

Figure 3: Color Representation for Curves

Part21 Example:

```
#87=EDGE_CURVE('',#28,#24,#66,.T.);
#242=DRAUGHTING_PRE_DEFINED_COLOUR('yellow');
#243=DRAUGHTING_PRE_DEFINED_CURVE_FONT('continuous');
#244=CURVE_STYLE('',#243,POSITIVE_LENGTH_MEASURE(1.0),#242);
#245=PRESENTATION_STYLE_ASSIGNMENT((#244));
#246=STYLED_ITEM('',(#245),#87);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268),#269);
```

4.2.3 Pre-Defined Colors

Table 2 below lists the pre-defined colors along with the RGB values that shall be assumed for them:

Color	RGB
'black'	0.0, 0.0, 0.0
'white'	1.0, 1.0, 1.0
'red'	1.0, 0.0, 0.0
'green'	0.0, 1.0, 0.0
'blue'	0.0, 0.0, 1.0
'yellow'	1.0, 1.0, 0.0
'cyan'	0.0, 1.0, 1.0
'magenta'	1.0, 0.0, 1.0

Table 2: RGB Values for Pre-Defined Colors

4.2.4 Transparency and Reflectance for Surfaces

In addition to a plain color, the appearance of a surface on screen can be enhanced by adding transparency and reflection characteristics as rendering options. This can be done instead of or in addition to the color instantiation as shown in section 4.2.2.

In general, whenever transparency or reflectance is defined, it is recommended to add an instance of `surface_style_rendering_with_properties` for the definition of a plain color to the `surface_style_fill_area` in Figure 2. This allows the realistic visualization of surfaces with properties which determine transparency and reflection characteristics, while maintaining compatibility with systems not supporting the capability.

Note: This instantiation currently violates MDGPR's where rule WR8, which excludes `surface_style_rendering` from being used as a `surface_side_style`. BugZilla #8179 has been raised to remove MDGPR.wr8 from AP242.

If a pattern (e.g. hatch) shall be applied as fill style, the two surface sides styled can be used simultaneously. In this case, the same `colour` shall be used by both styles, and the transparency shall be defined on the same surface side(s) the fill area style is defined on. Though it is legal to define more than one style, older systems might support only one style.

In addition to the `surface_colour`, the entity type `surface_style_rendering_with_properties` has the following attributes:

- `rendering_method`: specifies the method which shall be used for the shading of surfaces. This is an enumeration type with the following defined values:
  - `constant_shading`: a reflectance calculation is performed for each facet of the approximated surface to produce one reflected color per facet. The point on the facet used in the calculation is implementation-dependent. The color used in the reflectance calculation is the `surface_colour` specified in the relevant `surface_style_rendering` entity.
  - `colour_shading`: a reflectance calculation is performed at each vertex of each facet of the approximated product shape, using the `surface_colour` and the surface normals in the vertices. The resulting reflected colors are interpolated linearly across each facet.
  - `dot_shading`: any dot products needed by the reflectance equation are calculated from surface normals at a set of positions on the surface. These dot products are interpolated linearly across the surface. The reflectance calculation is performed at each interpolated position of the surface to produce a reflected color based on the interpolated dot products and the `surface_colour` of the relevant `surface_style_rendering` entity.
  - `normal_shading`: the surface normals are interpolated linearly across the surface. The reflectance calculation is performed at each interpolated position of the surface to produce a reflected color based on the interpolated surface normal and the `surface_colour` of the relevant `surface_style_rendering` entity.
- `properties`: This is a set of one or both rendering properties (transparency and reflectance), i.e. it is allowed to define only transparency, only the reflection characteristics, or both. For `surface_style_reflectance_ambient`, subtypes are defined which also allow for defining the diffuse and specular parts of the reflectance behavior of a surface.

Figure 4 illustrates the entity structure to define a transparent color for a surface:

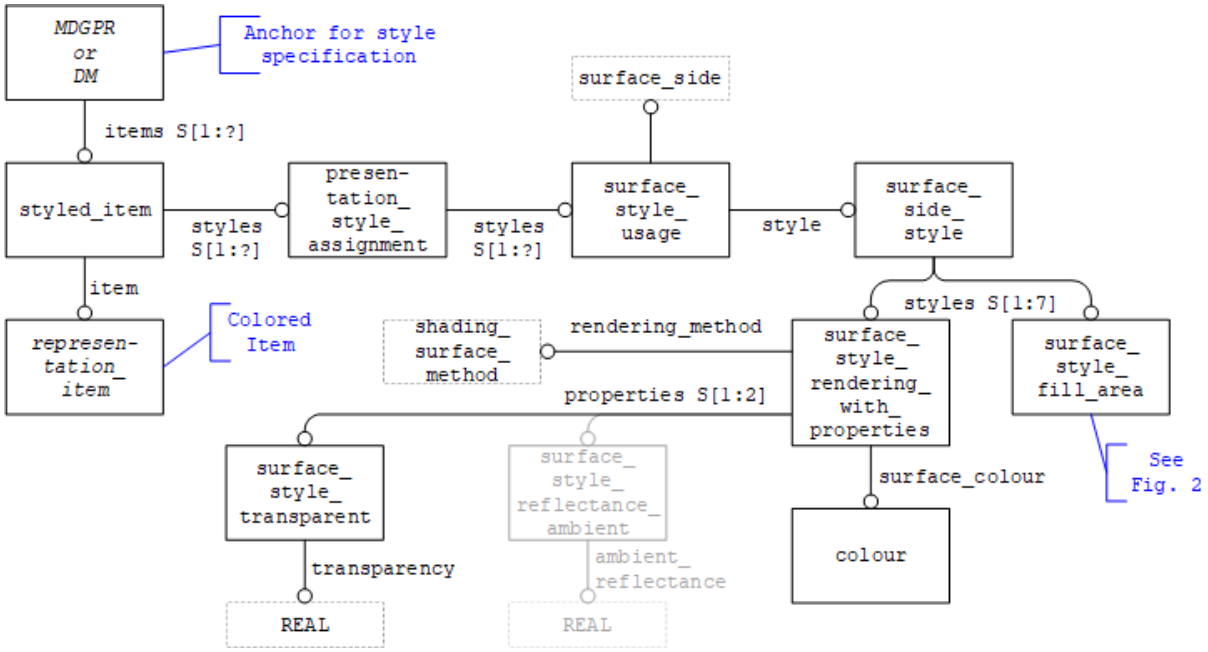

Figure 4: Definition of Transparent Color for a Surface

Pre-processor recommendation:

- The `surface_style_fill_area` and the `surface_style_rendering_with_-properties` must both use the same color.

Post-processor recommendation:

- Only one of the two surface styles shall be evaluated on import.
  - If the importing system supports `surface_style_rendering_with_-properties`, ignore the `surface_style_fill_area`.
  - If `surface_style_rendering_with_properties` is not supported, then evaluate the `surface_style_fill_area`.

4.2.5 Invisibility

Invisibility is a capability that will allow hiding of model elements. It is basically handled in the same way as any simple style (e.g. colors). Under the boundary conditions given in section 4.1 above, the usage convention is quite simple: Unless invisibility is present, all styled elements are visible.

The structure for invisibility is very simple: an instance of `invisibility` will be linked to the `styled_item` shown on the left-hand side of Figures 2-4:

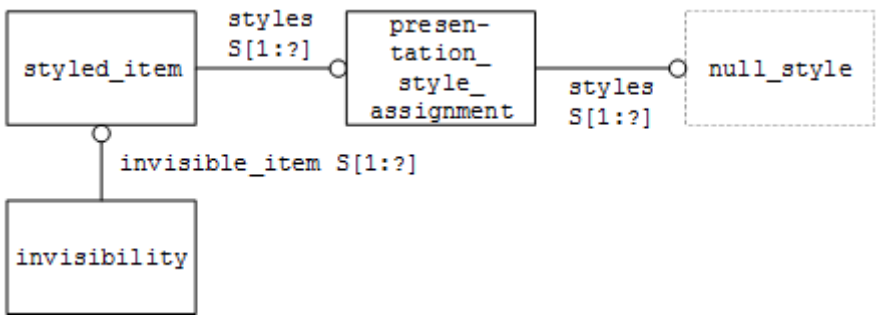

Figure 5: Structure of the Invisibility Assignment

In order to make sure the appearance of the component is not affected in systems not supporting invisibility, the `styled_item` shall reference the style originally assigned to the component, i.e. a `null_style` or a specific style as shown in Figures 2-4.

4.3 Overriding Styles

If one particular aspect of a model – e.g., one face of a solid – shall be styled differently, there is no need to break down the styles for each individual element. The inheritance order defined in 4.2.1 can still be used, and then the style for the element in question can be overridden with a new style.

The sections below define this for color, but it is applicable for the other styling options defined above as well.

4.3.1 Overriding Surface Colors

For coloring of a surface lying on a solid, the color of the solid is overridden. This happens by using an instance of `over_riding_styled_item`. Only portions of shape redefined by `over_riding_styled_item` will be affected. Portions not redefined by the overriding style shall be handled as already defined

Part21 Example:

```
#150=ADVANCED_FACE('',(#144),#149,.T.);
#247=DRAUGHTING_PRE_DEFINED_COLOUR('magenta');
#248=FILL_AREA_STYLE_COLOUR('',#247);
#249=FILL_AREA_STYLE('',(#248));
#250=SURFACE_STYLE_FILL_AREA(#249);
#251=SURFACE_SIDE_STYLE('',(#250));
#252=SURFACE_STYLE_USAGE(.BOTH.,#251);
#253=PRESENTATION_STYLE_ASSIGNMENT((#252));
#254=OVER_RIDING_STYLED_ITEM('',(#253),#150,#233);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268,#273),#269
);
```

Figure 6 illustrates the entity structure to define an overriding surface color.

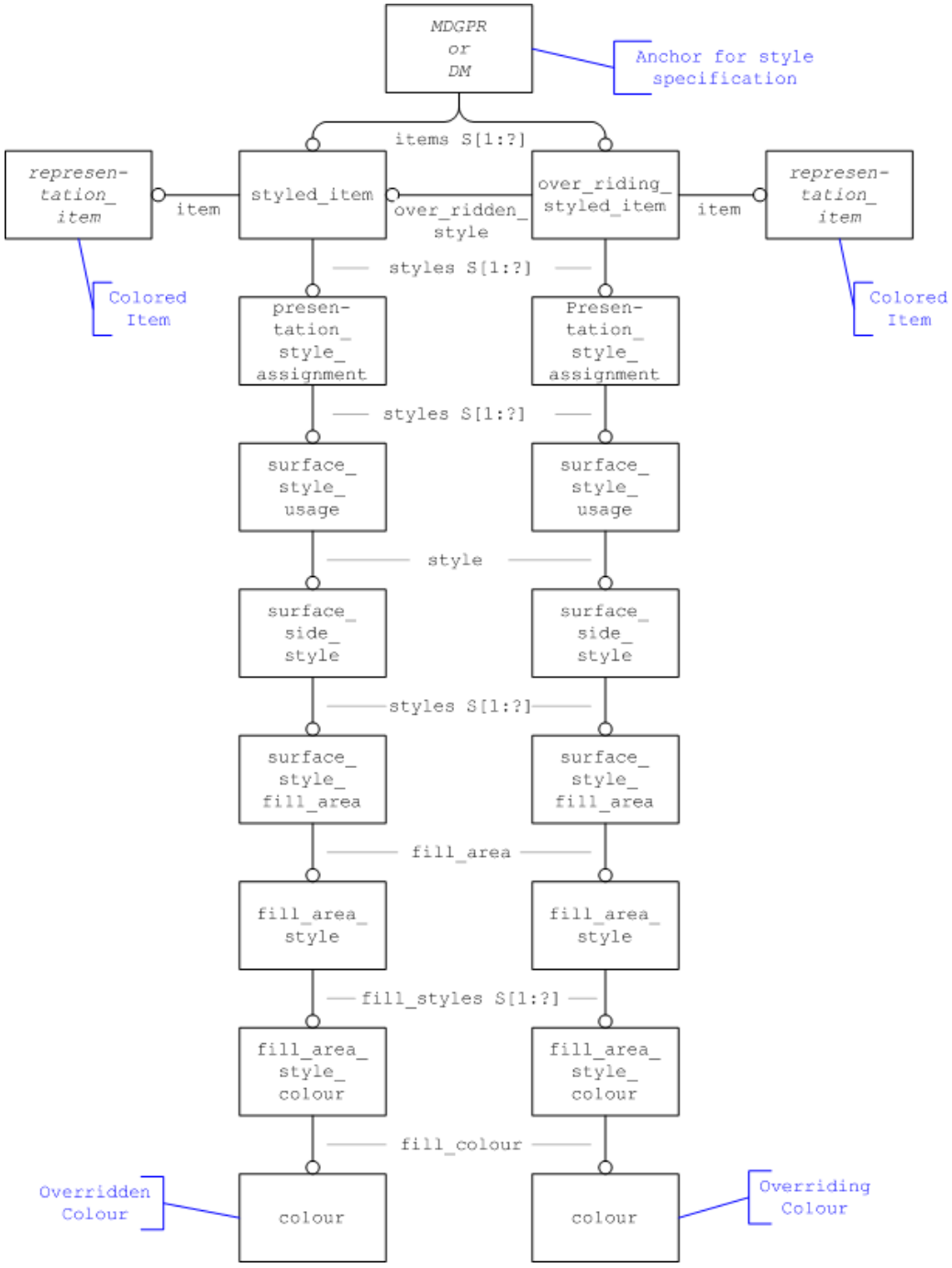

Figure 6: Representation of Overriding Surface Color

4.3.2 Overriding Edge Colors

For coloring of an edge, the color of the solid or surface is overridden. This happens by using an instance of `over_riding_styled_item`.

Note that edge colors are applied by a `curve_style` rather than a `surface_style`. This leads to a situation where the `surface_style` of the solid or surface containing the edge could be overridden by a `curve_style` for the edge in question.

Only portions of shape redefined by `over_riding_styled_item` will be affected. Portions not redefined by the overriding style shall be handled as already defined.

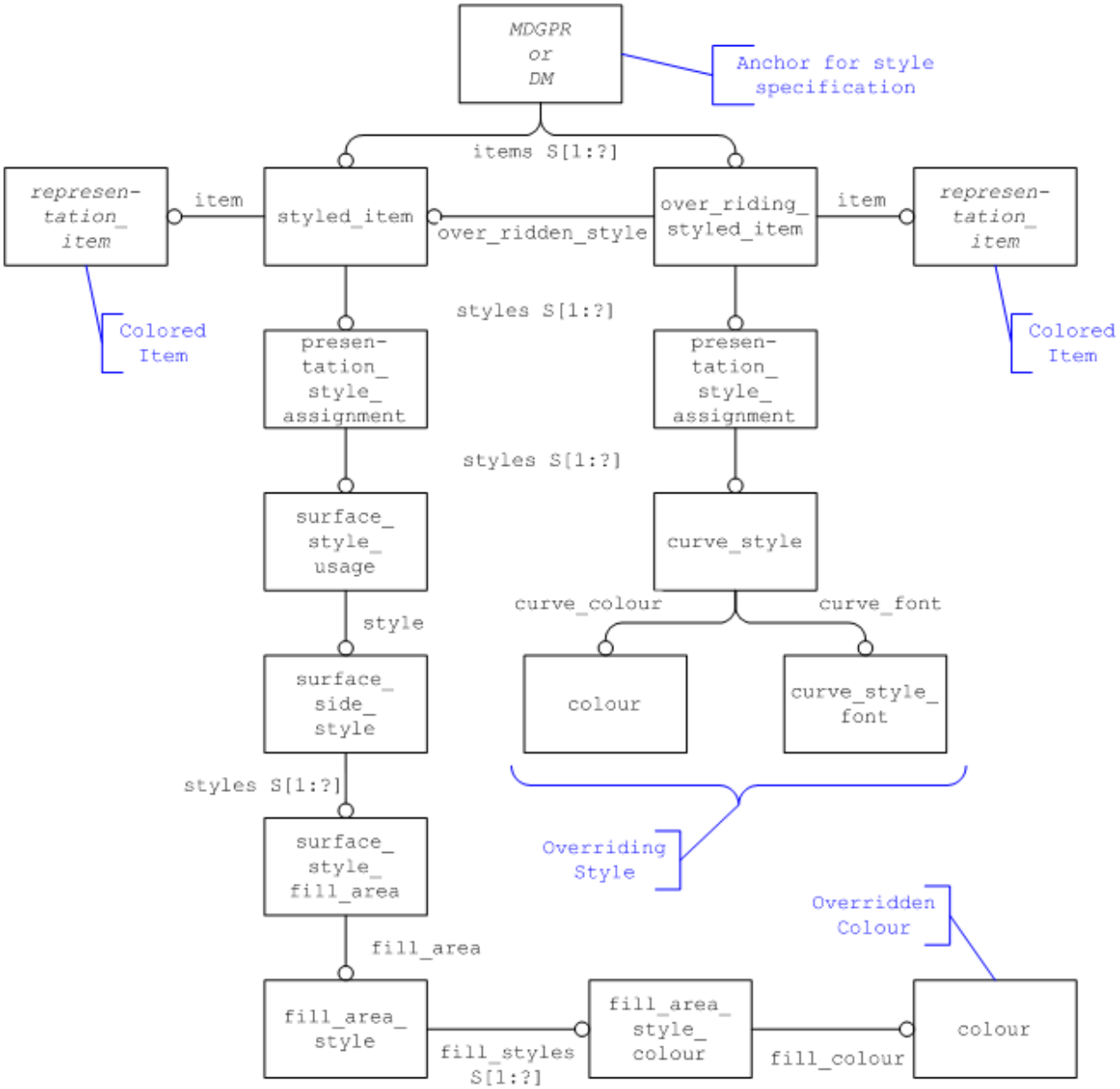

Figure 7: Representation of Overriding Edge Color

Part21 Example:

```
#87=EDGE_CURVE('',#28,#24,#66,.T.);
#242=DRAUGHTING_PRE_DEFINED_COLOUR('yellow');
#243=DRAUGHTING_PRE_DEFINED_CURVE_FONT('continuous');
#244=CURVE_STYLE('',#243,POSITIVE_LENGTH_MEASURE(1.0),#242);
#245=PRESENTATION_STYLE_ASSIGNMENT((#244));
#246=OVER_RIDING_STYLED_ITEM('',(#245),#87,#233);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268),#269);
```

4.4 Point Style & Color

In several scenarios it is desired to present particular points in the model on the screen, e.g. to illustrate weld spots or inspection points. A point by itself doesn't have any geometric extent, but in CAD systems such points are typically presented using a pre-defined marker, such as a cross or a circle. This marker can also have a color.

Figure 8 below illustrates the structure to define point style and color:

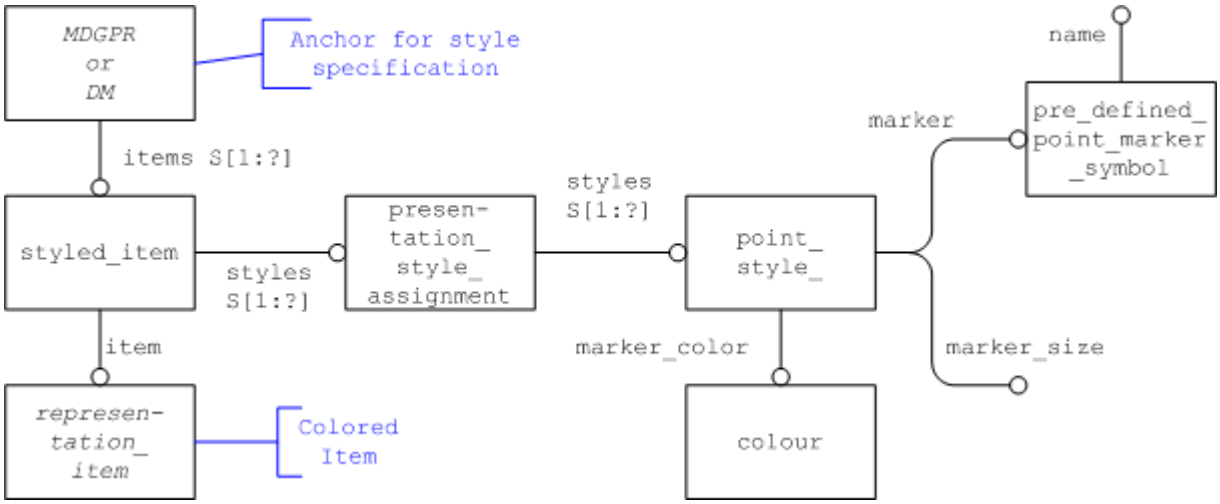

Figure 8: Definition of Point Style and Color

The `representation_item` in the lower left-hand corner of Figure 8 is typically a `cartesian_point` if a particular data point shall be presented. However, point style and color can also be assigned to curves and surfaces, as explained by the following quote from the definition of `presentation_style_assignment` in Part 46:

If a line is given a style which is a curve style, it shall appear. If a line is given both curve and point style, but the curve and its related cartesian points shall appear.

The marker size is typically given as a `positive_length_measure`.

For the definition of the marker itself, the entity type `pre_defined_point_marker_-symbol` shall be used (see Figure 8). It allows defining the following marker shapes by using the corresponding string as value for the name attribute:

Marker	Name value
*	'asterisk'
○	'circle'
●	'dot'
+	'plus'
□	'square'
Δ	'triangle'
X	'x'

Table 3: Pre-defined Point Marker Symbols

Part21 Example:

```
#44=CARTESIAN_POINT('Reference Point',(17.789,-11.092,26.287));
#322=PRE_DEFINED_MARKER('circle');
#323=DRAUGHTING_PRE_DEFINED_COLOUR('blue');
#324=POINT_STYLE(' ',#322,POSITIVE_LENGTH_MEASURE(2.),#323);
#325=PRESENTATION_STYLE_ASSIGNMENT((#324));
#326=STYLED_ITEM('',(#325),#44);
#327=DRAUGHTING_MODEL('#327',(#212,#243,#278,#326,#411 #504),#69);
```

4.5 Styling of Axis Placements

Depending on the use case, it is important to display coordinate systems defined in the model to the user, as they may carry process-relevant information such as tool targets etc. Other coordinate systems may be less relevant and hence preferred to be hidden, in order to avoid overloading the model display.

The STEP standard does not explicitly define specific styles for coordinate systems. Such a style definition could be very simple (one color), or it could be very detailed (different colors and curve styles for each axis, shape of the arrows, lengths of the axes etc.). This was discussed in BugZilla in #3526. It was agreed that the only meaningful style definition, if any, is to define one color for the entire coordinate system.

The main requirement is to define whether a coordinate system, represented by an `axis2_placement_3d`, shall be displayed to the user or not. This is done by applying the convention given in section 4.1:

- If there is no `styled_item` referencing the `axis2_placement_3d`, it shall not be displayed at all.
- If there is a `styled_item` (with a `null_style`) referencing the `axis2_placement_3d`, it shall be displayed to the user according to the target system's default settings.

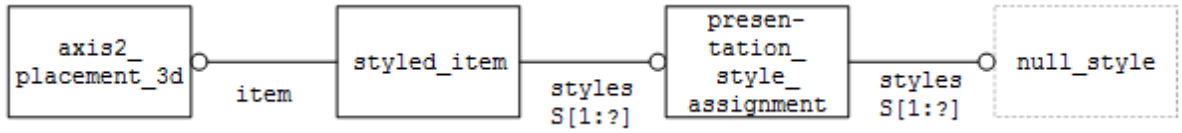

Figure 9: Displaying an Axis Placement

In order to support the requirement to define a single color for the entire placement, it was agreed that no new entity type is needed. An instance of `curve_style` with only the `curve_colour` defined, fulfills the requirement, as the intention of its use in this context can be easily deduced.

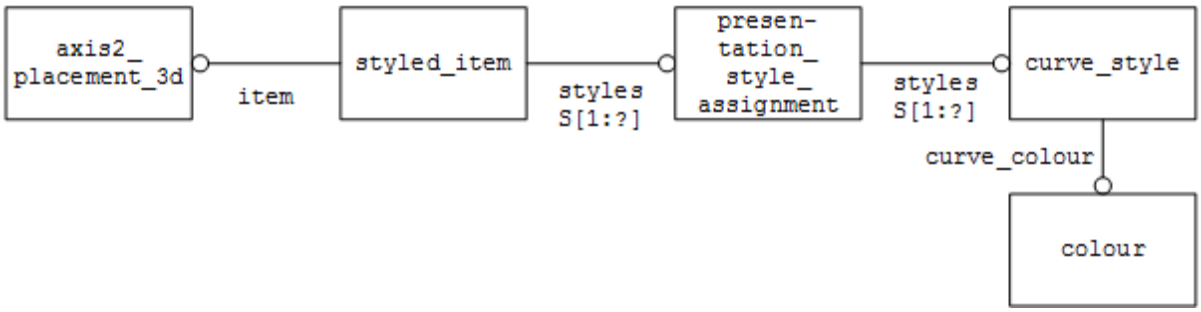

Figure 10: Coloring an Axis Placement

5 Assembly Component Instance Styling

The scope is the assignment of new (overriding) styles to individual instances of an assembly component, in order to emphasize certain parts in a given context. The style assigned is either a new color or an “invisibility” tag, which will declare the respective component as hidden.

5.1 Linking the Style to the Component Instance

The main item of interest in the context of assembly component instance styling is the identification of the correct instance.

Note: Until version 1.2 of this document, this was done using NAUO and SHUO, i.e. on the product assembly, to which then an empty representation was attached that served as anchor for the component instance style. In version 1.3 of this document, this has been replaced with a more elegant approach that works on the geometric assembly.

The core entity here is `context_dependent_over_riding_styled_item` (`CDORSI`), which is contained in AP203e2, AP214e3, and AP242. It is a subtype of `over_riding_styled_item` as introduced in sections 4.3 and 4.3.2 above, and has an additional attribute `style_context`, which is a list of, in this case, `representation_relationships`. This list is filled top-down, and thus unambiguously identifies the instance of the component the over-riding style shall be applied to. Figure 11 below gives an illustration based on the well-known AS1 example:

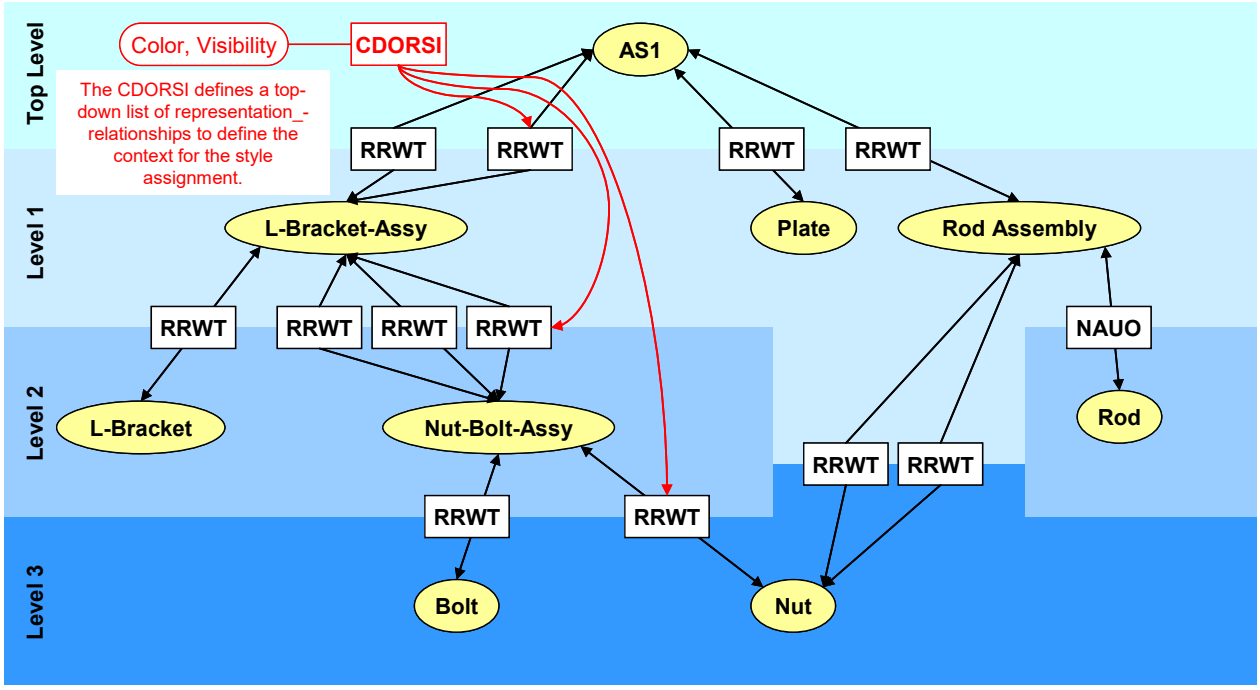

Figure 11: Identification of the Instance to be styled

The `CDORSI` refers to three entities to assign the style in the intended context:

- `item`: the item to be styled, e.g. the `shape_representation` of the component.
- `over_ridden_style`: The original style that was assigned to the component itself. Note that in the case of an assembly, this style may be assigned at a higher level in the assembly structure, from where it by definition applies to all sub-assemblies and components.
- `style_context`: the top-down list of `representation_relationships` that unambiguously define the path from the root node to the target leaf node instance.

The diagram in Figure 12 below illustrates the implementation structure for the case where one of the two Nuts of the Rod Assembly in AS1 shall be styled in a way different from the default style. This way, all other instances of Nut will be displayed in the target system's default color, while the specified instance will be shown in e.g. green.

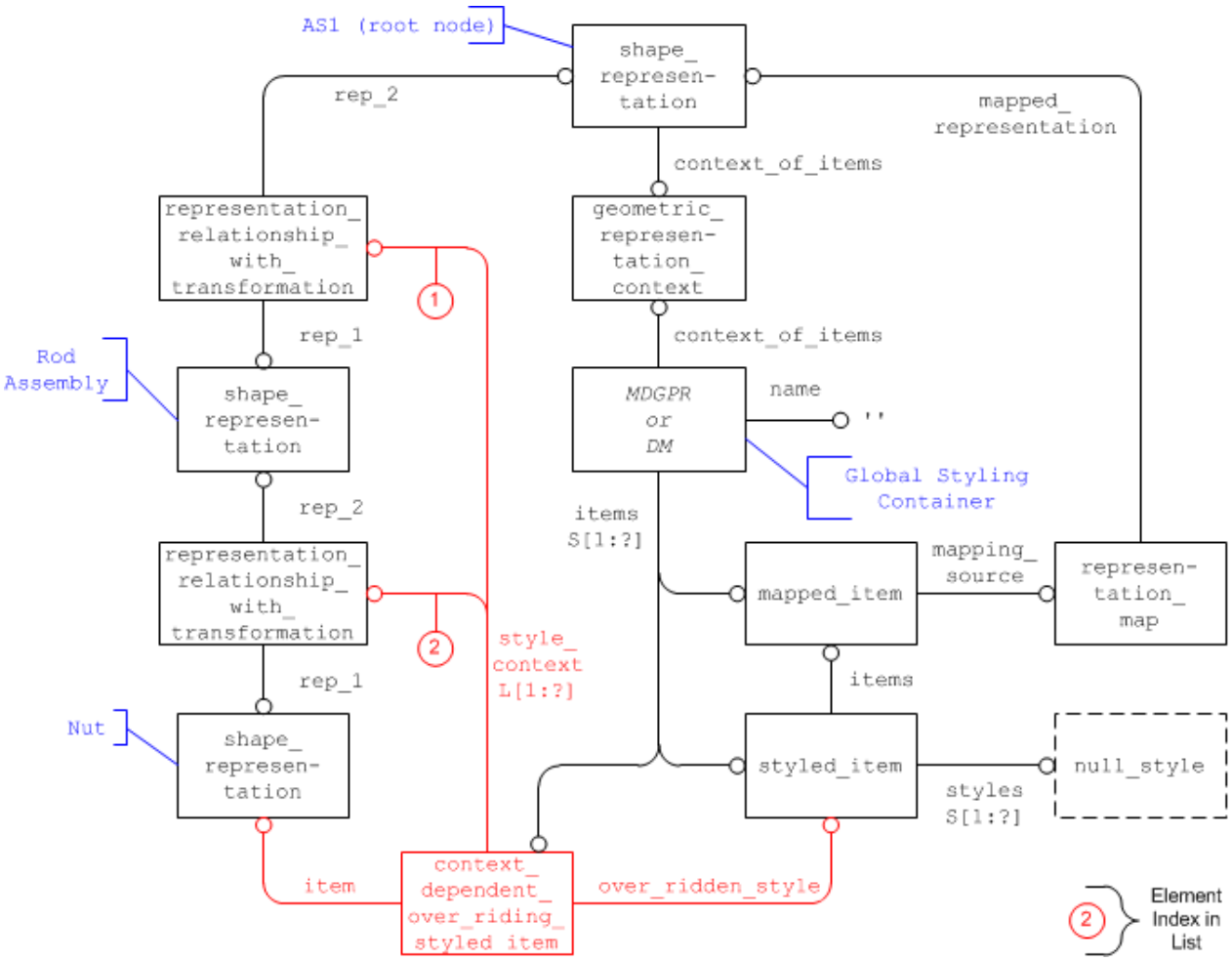

Figure 12: Styling an instance of a component in an assembly

The capability to style certain instances of components in an assembly in a different way can be extended even further by overriding the `CDORSI` itself with another instance `CDORSI`. Building on the example given in Figure 12 above, it can now not only be defined that while all instances of Nut are being displayed in the target system's default style one particular instance of Nut is shown in green; it can now be defined in addition that one face of that particular instance shall be red. This extension is illustrated in Figure 13 below:

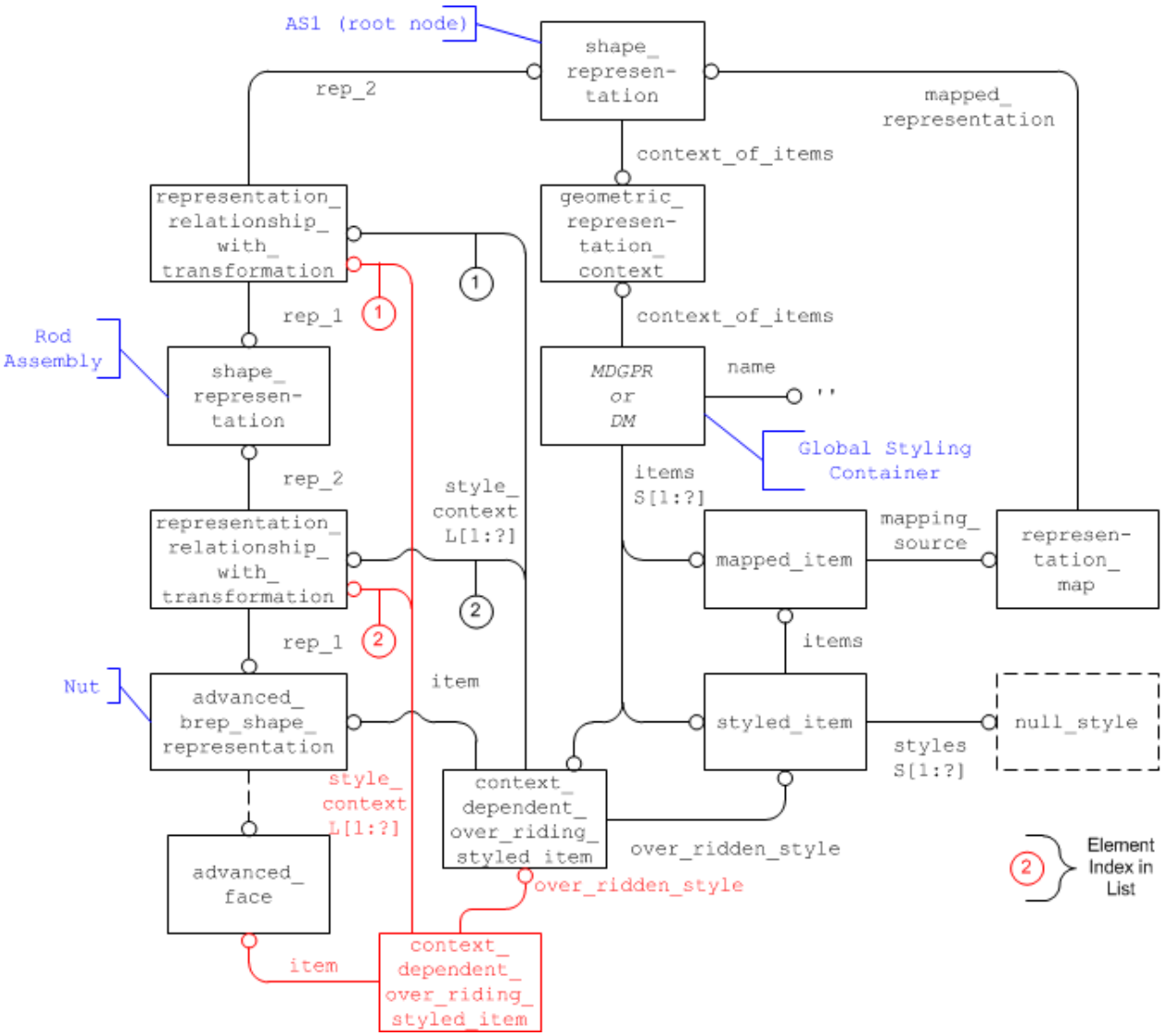

Figure 13: Styling a portion of an instance of a component in an assembly

5.2 Applicable Styles

The two styling characteristics described in this document that can be used to distinguish one instance of a component from its other instances are color and visibility.

5.2.1 Color / Transparency

The assignment of a new (transparent) color happens in the same way already described in sections 4.2.2 and 4.2.4 above. The `CDORSI` references in its set of `styles` an instance of `presentation_style_assignment`, which defines the new color for the component in the given context, see Figure 2 and Figure 4 respectively.

The styles referenced by `CDORSI` will override any similar styles defined on the overridden `styled_item`. Any type of styles defined on the original `styled_item` but not included in the `CDORSI` are kept unchanged. For example a green dashed line could be styled to appear as red dashed line by just giving the color red in the `CDORSI` while the dashed line type is kept from the original style.

5.2.2 Invisibility

For invisibility, the definitions given in 4.2.5 apply here as well. In the context of assembly instance styling, it can be used to hide a specific instance of a component. The structure for this is again very simple: an instance of `invisibility` will be linked to the `CDORSI`.

Note: Invisibility applies only to the `styled item` it is immediately assigned to and will be reset by any overriding styles. This means that if an item that is styled as invisible gets assigned an overriding style, it will become visible again, unless the overriding style again has an instance of `invisibility` assigned to it. To make a previously invisible item visible again without changing its original appearance, apply an overriding style with no `invisibility` assigned and with a "null" style in its set of styles.

6 Layers

A layer is a general structure for the collection of geometric and annotation elements.

6.1 Assigning Elements to a Layer

Layers shall not be nested, i.e.; a layer cannot be put on another layer.

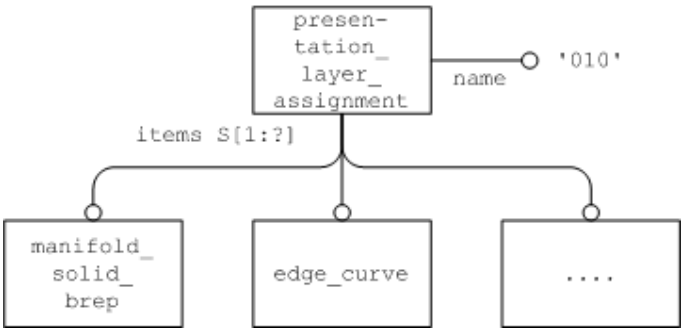

Figure 14: Layer Representation

The `presentation_layer_assignment` entity collects all items that are on the same layer in its assigned `items` attribute. These items are `representation_items`.

Part21 Example:

```
#225=PRESENTATION_LAYER_ASSIGNMENT('010','layer 010',(#206));
#206=SHELL_BASED_SURFACE_MODEL('#206',(#205));
```

Note that this ignores the informal proposition on `layered_item` provided in Part46. This approach was chosen by the Implementor Forum in order to reduce exchange file size by removing the need for `layered_items` to be `styled_items`, although a STEP file which applies layers through `styled_items` is not deemed to be invalid.

6.2 Layer Naming Recommendations

As various CAD systems have different mechanisms for organizing layers and the elements on them, the following recommendations are given to ensure interoperability

Pre-processor Recommendations:

- Create only one instance of `presentation_layer_assignment` per layer, and not for each individual element assigned to the layer.
- Do not use empty layer names. Some systems ignore layers with empty names on import.
- Use unique names for layers if the elements are expected to behave differently, e.g. regarding their visibility.

Post-processor Recommendations:

- If multiple layers with identical names are encountered on import, they shall be merged into a single layer in the target system.

6.3 Layer Styling

In addition, a whole layer can be set to invisible, using an instance of `invisibility` referencing the `presentation_layer_assignment`.

Note that even though all unstyled items are considered to be invisible, in the case of intended invisibility the invisible objects shall be declared explicitly as invisible. This may happen directly for each object or indirectly via declaring the layer containing the objects as invisible.

7 Groups

A group is an organizational structure that allows the grouping of specific elements in the model together. Relationships can be defined between groups to create a group hierarchy.

7.1 Assigning Elements to a Group

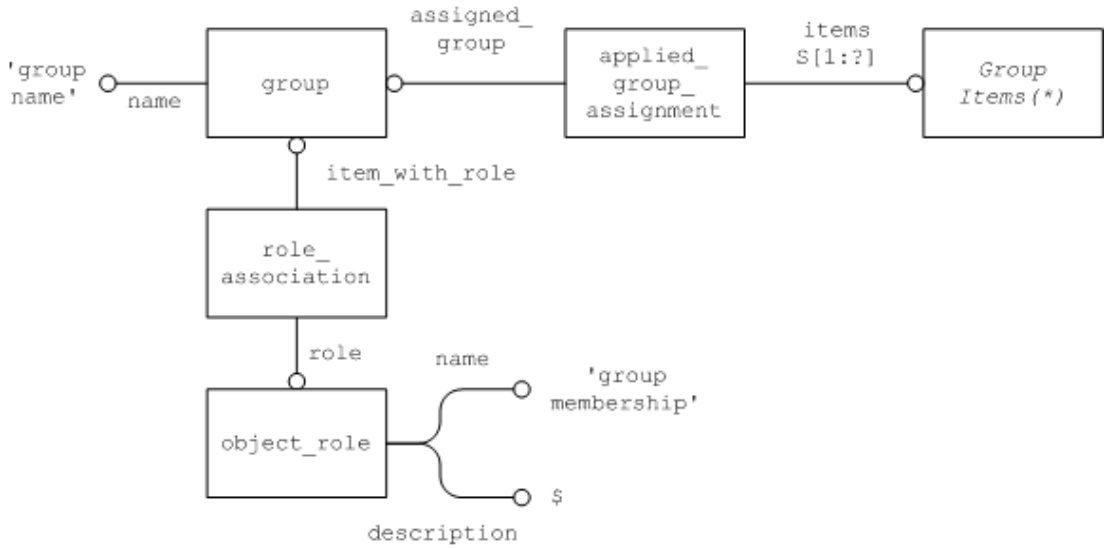

Figure 15: Group Assignment Representation

(*) **Note:** Even though `applied_group_assignment` is contained in both AP203e2 and AP214, the select types for `group_item` (AP214) and `groupable_item` (AP203e2) are different. AP203e2 allows more entity types to be grouped together. In AP214, a where rule further limits the groupable items to `geometric_representation_items` and `shape_aspects`.

As long as no further business requirements are known, the items in a group shall be limited to the AP214 definition.

7.2 Defining Group Relationships

Groups may be used to define other groups, and thus to create a group hierarchy.

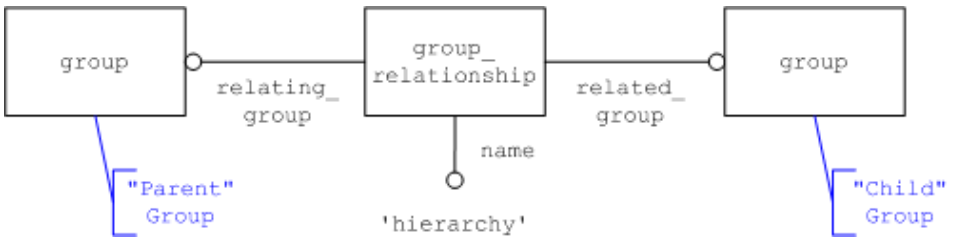

Figure 16: Group Relationship Representation

8 CATIA Geometric Sets

Geometric Sets are a model structuring mechanism that is specific to Dassault Systèmes' CATIA V5 and 3DEXPERIENCE CAD systems. It subdivides all constituent elements of a model into sets, such that each model element belongs to exactly one Geometric Set. The sets can be nested, i.e. a Geometric Set may contain another Geometric Set.

In the example below, the Geometric Set "GS1" contains "Point.1" and anther Geometric Set, "GS.2". This then contains "Point.2".

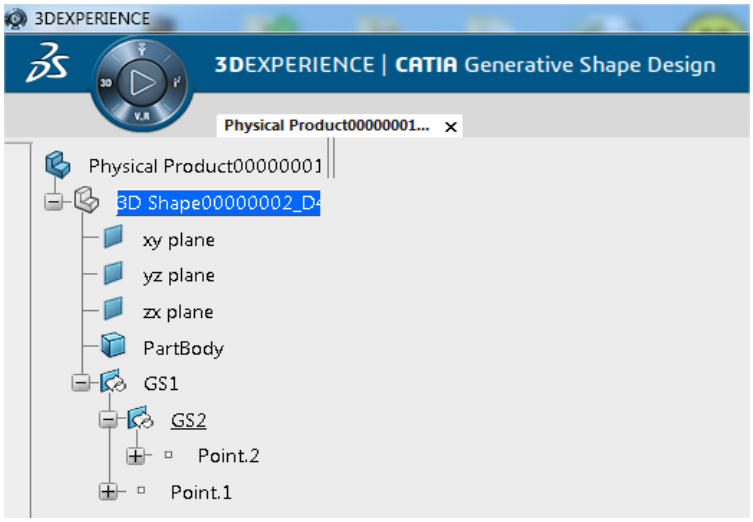

Figure 17: Geometric Set Example

While other CAD systems may have their own structuring mechanisms, none of them are similar or mappable to the Geometric Sets.

However, by design, the Geometric Sets can carry process-relevant information that needs to be preserved for roundtrip scenarios (e.g., CATIA – STEP – CATIA), e.g. for long-term archiving of digital design data. Hence, the requirement to preserve them in STEP. Though the Geometric Sets can be understood as a grouping mechanism, it was agreed not to use the Group functionality in STEP as described in section 0 above, in order to avoid ambiguity with the actual grouping mechanism that CATIA provides as well.

The following structure is recommended:

- Each Geometric Set will be represented as a `shape_aspect` with:
  - `shape_aspect.name` carrying the name of the Geometric Set, e.g. "GS1".
  - `shape_aspect.description` carrying the magic string "CATIA Geometric Set".
- The hierarchical structure will be represented using instances of `shape_aspect_relationship` with:
  - The `relating` attribute pointing to the parent set ("GS1" in the example above)
  - The `related` attribute pointing to the child set ("GS2" in the example above)
  - The `description` attribute carrying the magic string "CATIA Geometric Set".

Any target system other than CATIA V5 or 3DEXPERIENCE shall ignore any `shape_aspects` and `shape_aspect_relationships` with their description set to "CATIA Geometric Set".

CAx-IF testing determined Geometric Sets defined this way do not cause unwanted side-effects in other CAD systems.

To define the geometry assigned to the Geometric Set, the approach based on `shape_definition_representation` and `shape_representation` shall be used. This is useful in case the geometry shall be mapped into a saved view, and also helps to avoid conflicts with `shape_aspects` built using `item_identified_representation_usage` or `geometric_item_specific_usage` for PMI.

Note that in CATIA, Geometric Sets can be empty. This can be reflected in STEP, as it is not mandatory to define geometry for a `shape_aspect`.

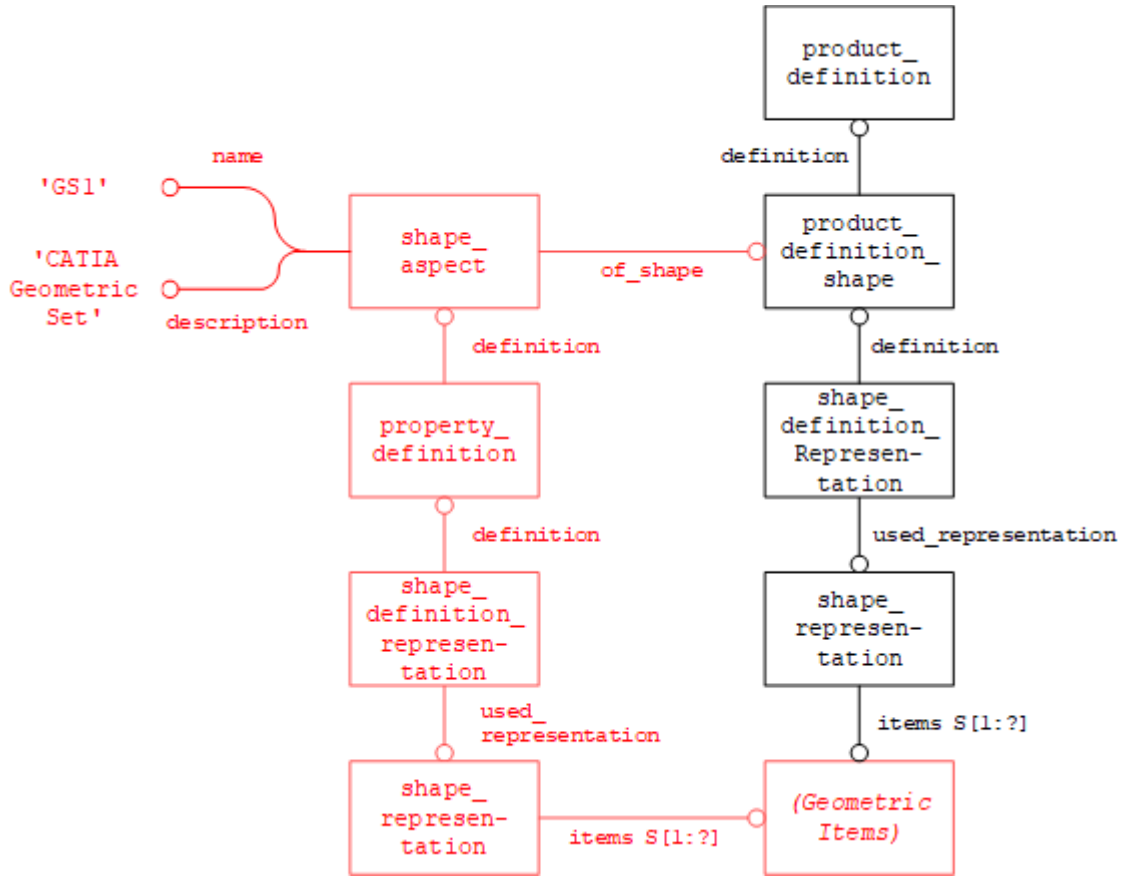

Figure 18: Defining Geometry for CATIA Geometric Sets

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 (2011) can be retrieved from:

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

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

B.3 AP242 Edition 1

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

B.4 AP242 Edition 2

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

- https://www.cax-if.de/documents/ap242ed2_mim_lf_v1.101.exp