Recommended Practices

for

Persistent IDs for Design Iteration and Downstream Exchange

Release 1.8

13 March 2026

CAx-IF

Jochen Boy
PROSTEP AG
jochen.boy@prostep.com

Technical

Asa Trainer
Consultant
agtrainer@comcast.net

Thomas Thurman
Consultant
thomas.r.thurman@imonmail.com

Table of Contents

List of Figures

Document History

This document is a new CAx-IF Recommended Practice and adds new constructs.

Release	Date	Change
0.1	2018-09-26	Initial release
0.2	2019-09-05	Rewritten to support either Part 21 Data Section (Attribute) approach and Part 21 E3 Anchor Section approach
0.3	2019-12-27	Added instance diagrams; limited Anchor Section discussion to refer to a future edition of the document
0.4	2020-01-14	Editorial revision
0.5	2020-05-28	Updated instantiation diagrams (Figures 2, 4, 5, 6)
0.6	2022-12-29	Updated to include UUID_ATTRIBUTE schema, versioning requirements, and iterative loading of imported data with UUIDs to emphasize the design iteration use case rather than the downstream-use use case;
0.7	2023-01-25	Additional tweaks before pre-release reviews including updated diagrams (Figures 2 through 7). Draft 0.7 out for review.
0.8	2023-02-01	Updated Mikael's information; Draft 0.8 out for review
0.9	2023-02-03	Minor changes to clarify use of UUID_attribute subtypes rather than abstract UUID_attribute supertype; Draft 0.9 out for CAx review
0.91	2023-05-18	Replaced all references to "GUID" with "UUID". This is based on ad hoc agreement with DMSC who are replacing "QPID" with "UUID" as well in QIF v4.0 (confirm this version with Larry or Curtis). Add highlights to the subtypes in section 4.2.2 to identify those subtypes that will limit scope and be the focus for this Rec Prac and the R52J and later Test Cases that reference it. Allow the publishing of UUIDs in either the Data Section form or the Anchor Section form. Either form is valid and post-processors should be able to read either type.
0.92	2023-06-07	Updated schema to push UUID_ATTRIBUTE under the Merkle tree; see Sections 4.2.2 and 5. Initial instantiation model is very similar to version 0.9 as the UUID_ATTRIBUTE inherits the 'uuid' identifier from uuid_leaf_node that has no other mandatory attributes.
0.93	2023-06-08	Replaced Figures 2 and 3. Changed P21 snippet in clause 4.2.1 to Example 1. Corrected record #1 in Example 1. Added notes to uuid_leaf_node, uuid_root_node, uuid_internal_node.
0.94	2023-06-28	Replaced Figures 2 through 8. Changed p21 snippet in Example 1. Updated EXPRESS in Clause 4.2.2: uuid_attribute_select, uuid_relationship_role,uuid_attribute, hash_based_v5_uuid_attribute, uuid_tree_node, uuid_leaf_node, uuid_internal_node, uuid_root_node, and uuid_context_role. Removed 4.4.2.1 id_attribute as we are no longer including id_attribute between uuid_attribute and the target. Updated Technical Authors info. Updated notes on Figures 4 through 8.Added link to trial schema in Annex A.
0.95	2023-11-20	Replaced Figures 2 and 3. Changed uuid_attribute.identified_item to: LIST [1:?] OF UNIQUE LIST[1:?] OF UNIQUE uuid_attribute_select. Note that Figure 2 includes a text note to this effect. Replaced uuid_leaf_node.data with a reference to uuid_attribute_select. Updated specification of file identification.
1.00	2023-11-28	All current changes accepted; DRAFT status changed to Released

Release	Date	Change
1.10	2024-06-15	Updated as follows – Section 4.2.2.3 - Deprecate HASH_BASED_V5_UUID_ATTRIBUTE; replaced with hash_function attribute on uuid_isolated_node and/or uuid_root_node. Section 4.2.2.6 - Added examples of UUID_RELATIONSHIP and UUID_RELATIONSHIP_ROLE including Figures 2a, 2b, 3a, and 3b. Remaining figures renumbered. Added Section 4.2.5 - PREPROCESSOR AND POSTPROCESSOR RECOMMENDATIONS FOR DOWNSTREAM CONSUMPTION to identify how geometric “features” are to be created and UUIDs assigned by pre-processors and how post-processors would handle these geometric features and their identifiers.
1.20	2024-12-10	Updated as follows – Acknowledgements – updated. Section 3 – Updated Documentation Identification for Release 1.20. Section 4.1 – Clarified the formation of UUIDs using the version 5 (Namespace UUID and Namestring-based UUIDs) approach and revised examples of the same. Section 4.2.1 – Clarified the requirement for revision/version information needed for iterative use of model data. Section 4.2.2 – Added new UUID_SET_ITEM, UUID_LIST_ITEM, and UUID_SET_ITEM_OR_UUID_LIST_ITEM_SELECT types. Section 4.2.2.6 – Updated the use of UUID_RELATIONSHIP when using UUID_SET_ITEM or UUID_LIST_ITEM examples. Sections 4.2.3 – Updated pre- and post-processor requirements to include new UUID_SET_ITEM and UUID_LIST_ITEM constructs. Section 5 – Updated Figure 4 with new UUID_SET_ITEM and UUID_LIST_ITEM constructs. Section A.4 – Updated schema name, description, and URL.
1.30	2025-07-07	Updates as follows – Section 3 – Updated Documentation Identification for Release 1.30. Section 4.2.2 – `vertex_point` has been removed as a target for UUID assignment based on discussion among CAx-IF vendors and organizers Section A4 – Revised expected timing of AP242 Edition 4 FDIS to Q3-Q4 CY 2025
1.40	2025-07-30	Updates as follows – Section 3 – Updated Documentation Identification for Release 1.40. Section 4.1 – Footnote added Section 4.2.3 – Paragraphs added about computing and identifying UUIDs assigned locally vs. those received as reference from external sources and also about having mechanisms to display UUID data to application users or to query UUID data by API-based methods in those applications.
1.50	2025-09-02	Updated examples in Section 4.2.2.6.2 to correct syntax.

Release	Date	Change
1.60	2025-10-15	Updated Section 3 to reference the current version of this Recommended Practice. Added Section 4.1, Permitted and disallowed uses of PIDs. All subsequent Subsections of clause 4 become 4.2, 4.2.1, etc. Change notes below also adjusted to follow the new numbering scheme. Updated Section 4.3.2 (was 4.2.2) to include footnotes on `shape_aspect` to clarify that UUID assignment for `datum`s shall be on the `datum_feature` or its subtypes, not on its related `datum` entity. Updated Section 4.3.5 (was 4.2.5) and its example to use `uuid_set_item`. Updated Section 5, Figure 8. Split Figure into two separate figures – Figure 8a using `uuid_set_item` Figure 8b using `uuid_list_item`. Updated Appendix A Schema URLs and added a URL to complete entity list for UUID referencing.
1.70	2025-10-29	Updated Section 3 to reference the current version of this Recommended Practice. Corrected and clarified compliance requirements in several sections. Corrected 2 Figure numbers. Ready for Release.
1.80	2026-03-13	Added Section 4.3.5.1 - System-specific treatment of PIDs for split hole entities (for R57J and beyond); Added Section 4.3.5.2 – UUID support for SURFACE_TEXTURE/SURFACE_TEXTURE_PARAMETER via UUID_LIST_ITEM; Added Section 4.3.5.3 – System-specific treatment of PIDs for split planar face entities (for R57J and beyond), • Specific to Ram/Viktor's Creo disjoint face in HTC or PDC/STC cases [Brutus #155] • will need to research whether this will apply to other systems (NX, V5/3DX); Added Section 4.3.5.4 - Sneak peek at collection of "feature" entities with PIDs for BOF uses (mfg and quality contexts) for R58J and beyond. Also added Appendix B.

Acknowledgements

The authors also wish to thank Mr. Mikael Hedlind, Sandvik, Sweden, Rosemary Astheimer, NIST, Jan de Nijs, Lockheed Martin, Larry Maggiano, Mitutoyo America Corporation, and Ed Paff, ITI, for their input on requirements for data exchange between CAD systems for design iteration and and/or for their input on requirements for data exchange for downstream use and/or for development of sample test models for discussion on design iteration exchange and downstream consumption use cases.

This document is principally prepared by Asa G. Trainer dba Trainer Engineering Associates, Interoperability Consultants, using funds from PDES, Inc. The statements, findings, conclusions, and recommendations are those of the author(s) and do not necessarily reflect the views of PDES, Inc or its member companies.

Portions of this report were prepared by Thomas Thurman dba TRThurman Consuling using Federal funds under Award No. 70NANB22H217 from the National Institute of Standards and Technology (NIST), U.S. Department of Commerce. The statements, findings, conclusions, and recommendations are those of the author(s) and do not necessarily reflect the views of NIST or the U.S. Department of Commerce.

1 Introduction

Research into mechanisms for maintaining traceability of engineering product data during exchange has concluded that the introduction and tracking of persistent IDs in that product data are feasible. Achieving traceability of individual data elements during state changes in product design and product development process can provide immediate practical benefits for several use cases. In this Recommended Practices document, we will focus on two such use cases – Design Iteration and Downstream Exchange. Review and discussion by CAx-IF at the April 2019 meeting, concluded on limiting the testing scope to Downstream Exchange. Design Iteration will be covered in a future version. After limited participation for the existing Downstream-use use case in 2020 and 2021 test rounds, and increasing interest in the Design Iteration use case, a discussion at the September 2022 CAx-IF meeting concluded that focus should shift to the latter use case.

1.1 *Design Iteration*

A significant problem in design in terms of efficiency is the difficulty in referencing shared data between team members (teaming partners or OEMs and their suppliers) during design iteration. Once a product model recipient consumes a source model into their CAD system and references product model entities from that source model in their design, they become locked into that instance of the external data. When subsequent versions of that source data are received and consumed again, re-mapping the references by hand, to maintain associativity between the now updated source data and the existing target design data, is challenging at best and nearly impossible for any but the most trivial of models.

Over the years, several major CAD vendors as well as at least one independent interoperability vendor, have implemented the ability to perform this associative mapping and update process automatically (Reference PTC's "Associative Topology Bus" (patent), NX's "Associative Update", Dassault Systemes' "Topological Naming", Integration Guard, etc.). These implementations have been successful, to a greater or lesser extent, using customized direct translation methods or within the boundaries of the individual vendor's exchange ecosystem. Implementation across system boundaries poses challenges including differences between systems in how and when entities are modified during design changes as well as differences in how each CAD system identifies model elements and the permanence of the identifiers for elements during change. For example, some systems may transform a geometric entity internally while maintaining its identifier while others may simply remove and replace the entity and update its own internal references on the fly. Another challenge is subtle differences between mathematical approaches to entity modeling in each CAD system and the impact those mathematical differences have on how entities are mapped. A well-known example is the modeling of hole features where some systems model the hole as a single cylindrical surface and other map the hole as two half cylinders. This example of a one-to-many surface mapping requires solid bookkeeping of entity identifiers by the receiving system.

In short, the ability to retain associativity between source feature, geometry, topology, and or attribute data and similar target data that may have dependencies on (references to) those source elements are key requirement for rapid iteration between product versions, thus improving design efficiency and reducing product development cost.

1.2 *Downstream Exchange*

Passing of design data to downstream systems has similar needs to maintain associative references between, for example, manufacturing tool paths and the model entities they are derived from, or dimensional tolerances as planned and measured in metrology systems and those same dimensional tolerances as defined in the original design systems where the models were created. In addition to the challenges of controlling and managing change in downstream systems based on model state change in designs, there is a second and perhaps more burdensome requirement for downstream consumption of model data and that is traceability. Traceability is necessary from the original system of creation of data through all its uses and throughout its lifecycle in order to follow potential faults to their source - either design, manufacture, material, or process. This traceability is a necessary and required forensic tool to be used in legal proceedings and technical investigations where complex product or process failures may have resulted in injury or death. In particular, the manufacturing and metrology communities have long had procedures in place to tag and maintain identification of products and their components, constituent entities, and attributes, to maintain this traceability and ensure their ability to isolate and identify any suspect system characteristics or entities in the event of failure. The ability to rapidly trace through suspect systems may be critical to quickly rectifying dangerous problems in product designs or product development processes.

1.3 Maintenance of this Document

This document will be maintained by the CAx-IF and will cover the Part 21 based implementations of persistent IDs. In the current version, this document focuses on both the design iteration use case and the metrology use case.

2 Scope

The following are within the scope of this document:

- The generation and use of Universally Unique Identifiers (UUIDs) (see Section 0 below) for maintaining associativity of entities in iterative design.
- The generation and use of Universally Unique Identifiers (UUIDs) (see Section 0 below) for maintaining associativity of entities for traceability for downstream uses of the model.
- The use of Part 21 Data Section or Part 21 Ed 3 Anchor Section for assigning UUIDs either on initial publication of the STEP document or after the original publishing of the STEP document should be supported.

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.---Persistent IDs---1.8---2026-03-13`

It will appear in a STEP file as follows:

```
FILE_DESCRIPTION(('...','CAx-IF Rec.Pracs.---Persistent IDs---1.8---2026-
03-13',),'2;1');
```

4 Persistent IDs

A mechanism for generating uniformly consistent, cross-application entity IDs is required for the above processes to work. These IDs need to be unique to prevent clashes between entity identifiers. In the issue summary for ISO Jira Task BS10303-3834 written by Thomas Thurman (see also BRUTUS #23), the scope of such uniqueness was suggested as only being required within the context of a specific product. This might be considered sufficient if only the first use case – design iteration – was required. In the context of the second use case, however, particularly the metrology use case, persistent (permanent), universally unique identifiers are necessary and need to be applied to product (as well as product effectivity, i.e., serialized product artifacts, if they exist), and to individual semantic PMI entities in the product.

4.1 Permitted and Disallowed Uses of Persistent Identifiers

Persistent Identifiers (UUIDs) are only permitted to be provided for use as external identifiers. No use of a Persistent Identifier (UUID) for internal file reference is permitted. The use of a Persistent Identifier (UUID) is not permitted to alter the logical structure of the data set by establishing intra-data-section cross-references. The current version of this recommended practice deprecates using `id_attribute` and `aggregate_id_attribute` for providing UUID assignment to data.

4.2 Formulation of Identifiers (UUIDs)

Fortunately, persistent, universally unique identifiers have been in use in the information technology domain for a long time. Such a 'universally unique identifier' (UUID) is a 128-bit number used to identify information in computer systems, e.g., operating systems, databases, and communications processes. The term 'globally unique identifier' (GUID) is sometimes also used. UUIDs have been standardized by the Open Software Foundation (OSF) and are documented as part of ISO/IEC 11578:1996 "Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)" and more recently in ITU-T Rec. X.667 | ISO/IEC 9834-8:2005. Most computing platforms provide convenient support for generating them, and for parsing their textual representation. More detail about UUIDs can be found on Wikipedia. Within the engineering domain, such UUIDs or GUIDs are already in place and being used in the Industry Foundation Classes (IFC) format of the Building Information Model (BIM) and in the Quality Information Framework (QIF) standard for the Metrology domain. A recent agreement has been reached to standardize the nomenclature to UUID in this STEP recommended Practice as well as the latest release of DMSC's QIF v4.0 release.

In the above standard there are 5 possible versions of UUIDs:

- Version 1 UUIDs are generated from a time and a node id (usually the MAC address),
- Version 2 UUIDs are generated from an identifier (usually a group or user id), time, and a node id,
- Versions 3 and 5 produce deterministic UUIDs generated by hashing a namespace identifier and name,
- Version 4 UUIDs are generated using a random or pseudo-random number.

Currently, the QIF standard suggests the use of Version 4 UUIDs, i.e., via random number seed. Several tools are generating UUIDs using this UUID Version. Based on research, there is some desirability to be able to consistently reproduce a UUID from some namespace and name data. Versions 3 and 5 allow this but the algorithm of Version 3 has been deprecated in favor of Version 5. Construction of Namespace UUIDs and Namestring-based UUIDs are discussed below.

A Namespace UUID is the responsibility of the publishing organization and is supplied to the publishing application for generation of appropriate Namestring-based model UUIDs, as described later in this section. Though there has been some discussion of registering namespaces specifically for these engineering information exchange use cases, no definitive plans have been put in place. Until such time as a registry repository is formed, any organization that generates Namespace UUIDs for their business processes and applications must retain their own repository for generated namespace UUIDs.

Namespace UUIDs are always owned by the creating (publishing) application and the defined Namespace UUID is unique to the application and is static. It should be formed from the following concatenated STEP data –

- <organization.id>+<application.id>+<domain>
- <application.id> includes the application name and version identification
- <domain> is an identifying marker for the domain of the publishing application (for example, MBSE / PLM / CAD / CAM / CAI / etc) [use of this domain element is OPTIONAL]

Namestrings for Namestring-based UUIDs are constructed by the application and ***must uniquely identify the data object*** of interest, for example

- <product.id>+<unique.entity.id or unique.path/entity.index>
- ***Change in properties of a data object do not change the Namestring***
- ***Change in model version does not change a Namestring***

UUIDS for any data object must persist unchanged across applications (domains), enterprises, and over time.

See examples for each of these below.

A version 5 UUID for Namespace could be constructed, for example, from:

- A pre-defined namespace: uuid.NAMESPACE
  - For example: UUID('6ba7b810-9dad-11d1-80b4-00c04fd430c8')
- Namespace String (SHA-1 hash of string) composed of

  Organization.id: “my company or division, my organization”

  Application.id: “CAD system XYZ, version 2024-01234”

  Domain: “Product Quality”

The function **uuid.uuid5**[1] with arguments **(uuid.NAMESPACE, "my company or division, my organization CAD system XYZ, version 2024-01234, Product Quality")**

renders **uuid.NAMESPACE = ‘aa378c77-d030-5f0f-9cce-ddfdb81be968’**.

A version 5 UUID for a particular Feature Control Frame in a named model could be constructed, for example, from:

- A pre-defined namespace: uuid.NAMESPACE
  - From above example: UUID(‘aa378c77-d030-5f0f-9cce-ddfdb81be968’)
- Model Identification String (SHA-1 hash of string) composed of

  Product.id: “123456789-1”

[1] Most operating systems, scripting languages, programming languages, and database applications support UUID generation. In the example shown, Python’s uuid module generates the type 5 uuid via the function call uuid.uuid5(). Other examples for this purpose include the java.util.UUID class methods, the Linux system util-linux package function uuidgen, and the Bash script’s uuidgen command. For a detailed survey of UUIDs refer to „Beyond Randomness: A Detailed Study on UUID Standards, Data Integrity, and Identifier Design Across Storage Systems“, A. R. Sinha, IJIRCT ISSN 2454-5988, 2023

Type: “PMI Feature Control Frame”

Persistent ID: “ID 879819”

The function **uuid.uuid5**[1] with arguments **(uuid.NAMESPACE, "123456789-1, PMI Feature Control Frame, ID 879819")**

renders **uuid.Namestring = 8cab1aa3-0080-55a7-8181-6fbf4d831ca7'**

(for part number 123456789-1, entity 879819)

It is important to note that, if all hash string elements are the same, the UUID generator will generate the same UUID again.

Model Identification String (SHA-1 Hash String) is the responsibility of the owner (creator) application. The creator application should concatenate string elements including the following:

- a unique part number,
- a part instance serial number (if it exists),
- an entity type or full path from the product to the individual entity, and
- an internally maintained, unique persistent entity ID.

In the remainder of this recommended practice document, we will refer to the term UUID when describing these unique identifiers.

4.3 IDs in STEP

There are two valid approaches to storing UUIDs in STEP. The first is via an internal identifier within the context of the Part 21 data section. This method was originally suggested by data modelers and may make sense for the original publishers of STEP from the source CAx system and will be described in Section 4.3.2 below. The second is the storing of such identifiers not in the data section but rather within the Part 21 Edition 3 Anchor Section. This second method was once put forward primarily as an alternative method to add identifiers to a STEP file for newly appended data after its original publication or to allow addition of identifiers to legacy STEP data. Either of the above methods – Data Section or Anchor Section – are considered valid for initial publication by the source preprocessor or for appending data after initial publication.

4.3.1 Product Identification and Model Versioning to Support Iteration

Identification of product is necessary to support and product exchange activity whether for iteration between design systems or for iteration between design systems and downstream systems. For identification across systems, a `UUID_attribute` entity needs to be assigned to product. The `product_definition` and `product_definition_shape` entities are used for this purpose. An example of the connection between these two entities and a `UUID_attribute` entity (Section 4.3.2 below) is given in example 1 below:

Example 1

```
#1=v5_uuid_attribute  ('36  character  uuid  string',(#3));
/*identified_item is the second attribute in the attribute
list */

#2=product_definition('','',#5,#4);

#3=product_definition_shape(#2);

#4=product_definition_context(...);

#5=product_definition_formation_with_specified_source('',
'',#6,.NOT_KNOWN.);

#6=product(....);
```

To support iteration, use cases for either design iteration or iteration with downstream applications, some mechanism is needed, at the application level, to identify the change of state of the model from one iteration to the next. This is typically done by checking the model out of a PLM system, performing some modification on the design model, and checking that next model iteration back into the PLM system. This requirement also applies between revisions of the model, i.e., when major changes to the design are published for use within the extended enterprise. In either case (iteration or revision), a counter is incremented to identify the change in state of the model. Often this counter information is carried not only within the PLM system, but also within the model itself. In order to support iterative design and update during STEP exchanges, this iteration flag, i.e., the revision/version counter, also needs to be injected into the STEP file when created and also needs to be captured by any consuming STEP postprocessor. The format of this iteration flag is a string in form of <revision> or <revision letter>.<version number> or <revision letter>-<version number>, e.g. -, A, B, C, etc, or -, A.1, A.2, B.1, B.2, etc, or -, A-1, A-2, B-1, B-2, etc. The `product_definition` and `product_definition_formation` entities are used for this purpose. An example of the connection between these two entities and a `UUID_attribute` entity (Section 4.3.2 below) is given in example 2 below:

Example 2

```
#1=v5_uuid_attribute  ('36  character  uuid  string',(#2));
/*identified_item is the second attribute in the attribute
list */

#2=product_definition('','',#3,#4);

#3=product_definition_formation('A.1','',#5);

#4=product_definition_context(...);

#5=product(....);
```

An iteration/revision flag must be populated using the default value '-', if no other revision/version information is available.

4.3.2 Persistent ID (UUID) Entity Identification

The EXPRESS entities and attributes used to support the requirements of UUID entity identification and relationships between them are illustrated below (Figure 1, also refer to the schema diagram in Figure 2). They are included in AP242 Edition 4. In case of any discrepancies between these test schema entries and the published AP242 Edition 4 should be brought to the attention of the authors.

NOTE - Because Merkle Tree is still a topic of research, it will be held aside at least until Edition 5 while research on net change exchange continues.

Note that the list of types for `id_attribute_select` and `identification_item` given below is not an exhaustive list, i.e. it does not include all subtypes that may be referenced. Please refer to the complete list of all possible subtypes, generated by traversing subtype and select type structures in AP 242 Edition 4, as referenced in Appendix A.5. **Only those shown in bold** in the list below will be used for the purposes of entity identification for persistent ID usage at this time. An entry in either list below may be a supertype and this implies that any subtypes of that entity are also available for assignment with a UUID.

TYPE id_attribute_select = SELECT
(action,
address,
application_context,
ascribable_state_relationship,
dimensional_size, [**dimensional_size_with_path**]
geometric_tolerance,
group,
organizational_project,
product_category,
property_definition,
representation, [advanced_brep_shape_representation, **shape_representation**,]
shape_aspect[2],
shape_aspect_relationship, [**dimensional_location**]
topological_representation_item); [advanced_face, closed_shell, open_shell, edge_curve]
END_TYPE;

TYPE identification_item = SELECT
(
action,
application_context,
characterized_object,
characterized_object_relationship,
context_dependent_shape_representation,
derived_unit,
dimension_related_tolerance_zone_element,
dimensional_characteristic_representation,
dimensional_location,
founded_item,
geometric_tolerance_auxiliary_classification,
geometric_tolerance_relationship,
gps_filter,
gps_filtration_specification,
invisibility,
item_identified_representation_usage,

[2] For a `shape_aspect` of subtype `datum`, its UUID shall be assigned to the `datum_feature` entity (or one of its subtypes), not to its related `datum` entity.

```
limits_and_fits,
measure_qualification,
measure_with_unit,
named_unit,
plus_minus_tolerance,
representation_item,
representation_item_relationship,
runout_zone_orientation,
tolerance_value,
tolerance_zone_definition,
tolerance_zone_form,
   action_directive,
   action_directive_relationship,
   action_method,
   action_method_relationship,
   action_property,
   action_property_representation,
   action_relationship,
   address,
   alternate_product_relationship,
   alternative_solution_relationship,
   analysis_assignment,
   analysis_representation_context,
   applied_action_assignment,
   applied_action_method_assignment,
   applied_action_request_assignment,
   applied_approval_assignment,
   applied_certification_assignment,
   applied_classification_assignment_relationship,
   applied_contract_assignment,
   applied_description_text_assignment,
   applied_description_text_assignment_relationship,
   applied_document_reference,
   applied_document_usage_constraint_assignment,
   applied_effectivity_assignment,
   applied_event_occurrence_assignment,
   applied_external_identification_assignment,
   applied_external_identification_assignment_relationship,
   applied_identification_assignment,
   applied_ineffectivity_assignment,
   applied_organization_assignment,
   applied_organizational_project_assignment,
   applied_person_and_organization_assignment,
   applied_security_classification_assignment,
   applied_time_interval_assignment,
   applied_usage_right,
   approval,
   approval_relationship,
   approval_status,
   ascribable_state,
   ascribable_state_relationship,
   assembly_component_usage,
   assembly_component_usage_substitute,
   assignment_object_relationship,
   breakdown_element_realization,
   breakdown_of,
   certification,
   change_group,
   characterized_class,
   class,
   class_system,
   configuration_effectivity,
   configuration_item,
   configuration_item_relationship,
   contract,
   contract_relationship,
   date_and_time_assignment,
   date_assignment,
   degenerate_pcurve,
   dimensional_size,
   dimensional_size_with_path,
   directed_action_assignment,
   document_file,
   document_relationship,
   document_type,
   draughting_model,
   effectivity,
   effectivity_relationship,
   envelope,
   envelope_relationship,
   evaluated_characteristic,
   event_occurrence,
   event_occurrence_relationship,
   evidence,
   exclusive_product_concept_feature_category,
   executed_action,
   general_property,
   general_property_relationship,
   generic_property_relationship,
   group,
   group_relationship,
   identification_assignment_relationship,
   information_right,
   information_usage_right,
   interface_connection,
   interface_connector_as_planned,
   interface_connector_as_realized,
   interface_connector_definition,
   interface_connector_design,
   interface_connector_occurrence,
   interface_connector_version,
   interface_definition_connection,
   interface_definition_for,
   interface_specification_definition,
   interface_specification_version,
   link_motion_relationship,
   material_designation,
   material_designation_characterization,
   measure_representation_item,
   mechanical_design_geometric_presentation_representation,
   message_relationship,
   organization,
   organization_relationship,
   organizational_address,
   organizational_project,
   organizational_project_relationship,
   package_product_concept_feature,
   person,
   person_and_organization,
   person_and_organization_address,
   point_on_surface,
   presentation_area,
   process_operation,
   process_plan,
   product,
   product_category,
   product_class,
   product_concept,
   product_concept_context,
   product_concept_feature,
   product_concept_feature_category,
   product_concept_relationship,
   product_definition,
  product_definition_formation,
   product_definition_formation_relationship,
   product_definition_occurrence,
   product_definition_occurrence_reference,
   product_definition_relationship,
   product_definition_usage,
   product_definition_usage_relationship,
   product_group,
   product_group_membership,
   product_group_relationship,
   product_identification,
   product_process_plan,
   product_relationship,
   property_definition,
   property_definition_relationship,
   property_definition_representation,
   representation,
   representation_context,
   representation_relationship,
   requirement_assignment,
   requirement_for_action_resource,
   requirement_source,
   retention,
   rule_set,
   satisfies_requirement,
   security_classification,
   security_classification_level,
   shape_aspect[3],
   shape_aspect_relationship,
   shape_feature_definition,
   shape_feature_definition_relationship,
   shape_representation,
   state_definition_to_state_assignment_relationship,
   state_observed,
   state_observed_assignment,
   state_observed_relationship,
   state_type,
   state_type_assignment,
   state_type_relationship,
   structured_message,
   time_interval,
   time_interval_relationship,
   usage_association,
   validation,
   verification,
   verification_relationship,
   versioned_action_request,
   versioned_action_request_relationship);
END_TYPE;


  TYPE uuid_attribute_select = SELECT
    (id_attribute_select,
     identification_item);
  END_TYPE;

  TYPE uuid = STRING (36) FIXED;
  END_TYPE;

  TYPE uuid_list_item = LIST [1 : ?] OF UNIQUE LIST [1 : ?] OF UNIQUE
uuid_attribute_select;
  END_TYPE;

  TYPE uuid_relationship_role = ENUMERATION OF
    (SUPERSEDES,
     MERGE,
     SPLIT,
     DERIVE_FROM,
     SAME_AS,
     SIMILAR_TO);
  END_TYPE;

  TYPE uuid_set_item = SET [1 : ?] OF uuid_attribute_select;
```

[3] For a shape_aspect of subtype datum, its UUID shall be assigned to the datum_feature entity (or one of its subtypes), not to its related datum entity

```
  END_TYPE;

TYPE uuid_set_or_list_attribute_select = SELECT
   (uuid_list_item,
    uuid_set_item);
END_TYPE;

ENTITY uuid_attribute
  ABSTRACT SUPERTYPE OF(ONEOF(
    v5_uuid_attribute,
    v4_uuid_attribute)
    ANDOR uuid_attribute_with_approximate_geometric_location)
   identifier : uuid;
  identified_item : uuid_set_or_list_attribute_select;
 UNIQUE
  UR1 : identifier;
END_ENTITY;

  ENTITY v5_uuid_attribute
    SUBTYPE OF(uuid_attribute);
  END_ENTITY;

  ENTITY v4_uuid_attribute
    SUBTYPE OF(uuid_attribute);
  END_ENTITY;

  ENTITY hash_based_v5_uuid_attribute
    SUBTYPE OF(v5_uuid_attribute);
      hash_function       : STRING;
    WHERE
      WR1 : hash_function <> '';
  END_ENTITY;

  ENTITY uuid_attribute_with_approximate_location
    SUBTYPE OF(uuid_attribute);
      location_representation : shape_representation;
      approximate_location     : cartesian_point;
    WHERE
      WR1 : location_representation IN
using_representations(approximate_location);
  END_ENTITY;

*)
  ENTITY uuid_relationship;
      identifier : uuid;
      uuid_1     : uuid;
      uuid_2     : uuid;
      role       : uuid_relationship_role;
      tree_root  : OPTIONAL uuid_tree_root;
    UNIQUE
      UR1 : identifier;
    WHERE
      WR1 : uuid_1 <> uuid_2;;
```

```
        WR2 : uuid_1 <> identifier;;
        WR3 : identifier <> uuid_2;;
        wr4 : NOT ((parent_child) = role) OR EXISTS(tree_root);
    END_ENTITY;

    ENTITY uuid_provenance;
        identifier : uuid;
        content    : LIST [1:?] OF UNIQUE uuid_relationship;
      UNIQUE
        UR1 : identifier;
    END_ENTITY;

    ENTITY uuid_tree_node
      ABSTRACT
      SUPERTYPE OF (ONEOF(uuid_leaf_node, uuid_internal_node));
        identifier : uuid;
        node_2     : OPTIONAL uuid_tree_node;
        node_1     : OPTIONAL uuid_tree_node;
      WHERE
        WR1 : node_1 <> node_2;
    END_ENTITY;

  ENTITY uuid_leaf_node
      SUBTYPE OF(uuid_tree_node);
     data : uuid_attribute_select;
      DERIVE
        leaf_operand : STRING (1) FIXED := '0';
      WHERE
        WR1 : NOT (EXISTS (node_1) OR EXISTS(node_2));
        WR2 :
(SIZEOF(USEDIN(SELF,'AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF.
UUID_TREE_NODE.NODE_1')) = 1) AND
(SIZEOF(USEDIN(SELF,'AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF.
UUID_TREE_NODE.NODE_2')) = 1);  END_ENTITY;

    ENTITY uuid_internal_node
      SUBTYPE OF(uuid_tree_node);
      DERIVE
        internal_operand : STRING (1) FIXED := '1';
      WHERE
        WR1 : EXISTS(node_1) AND EXISTS(node_2);
        WR2 : (SIZEOF(USEDIN(SELF,
'AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF.UUID_TREE_NODE.NODE_
1'))=1) AND (SIZEOF(USEDIN(SELF,
'AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF.UUID_TREE_NODE.NODE_
2')) = 1);
    END_ENTITY;

    ENTITY uuid_root_node
      SUBTYPE OF(uuid_internal_node);
        hash_function : STRING;
      DERIVE
        root_operand  : STRING := '1';
    WHERE
```

```
          WR1 : SIZEOF(USED_IN(UUID_SCHEMA.UUID.TREE_NODE.NODE_1)) = 0;
          WR2 : SIZEOF(USED_IN(UUID_SCHEMA.UUID.TREE_NODE.NODE_2)) = 0;
          WR3 : EXISTS(SELF\uuid_tree_node.node_1) AND
    EXISTS(SELF\uuid_tree_node.node_2);
      END_ENTITY;
    (
      ENTITY uuid_context_role;
          identifier : uuid;
          role       : STRING;    UNIQUE      UR1 : identifier;
    WHERE
       WR1 : role <> '';  END_ENTITY;
```

Figure 1: EXPRESS Entities for Persistent IDs

4.3.2.1 UUID_ATTRIBUTE

The `uuid_attribute` entity is an extension from the `id_attribute` entity that represents UUID-specific identifier information. The `uuid_attribute` is abstract and only subtype of `uuid_attribute` must be populated, i.e. `v5_uuid_attribute` and `v4_uuid_attribute`, as appropriate. A `uuid_attribute` associates a UUID with an ordered or unordered collection of product data items. Only items specified by `id_attribute_select` or those specified by `identification_item`, shall be specified by `uuid_attribute`. In current trials, either the ordered or unordered collection is to be used, and it may contain one or more items. The ordered collection is specified by `uuid_list_item` and the unordered collection by `uuid_set_item`.

As a follow-on trial (timing to be determined), the merkle tree (`shape_aspect…advanced_face`, `shape_aspect…geometric_tolerance`, `shape_aspect…presentation geometry`) will be exchanged.

4.3.2.2 V5_UUID_ATTRIBUTE

The `v5_uuid_attribute` entity is a SUBTYPE OF uuid_attribute, and inherits the attributes from that ENTITY. A `v5_uuid_attribute` is a `uuid_attribute` that provides a UUID that conforms to version 5 of the relevant rfc. Version 5 UUIDs are generated based on a known namespace identifier and a name string that can be relied on between iterations of the product. The names string nor the namespace are provided. They are left to the implementor to manage internally (refer to Section 4.2 for details).

Note that there is an error in v5_uuid_attribute.wr1 related to character position of the version attribute. That will be corrected in ed5. The where rule shall be ignored during instance data validation.

4.3.2.3 HASH_BASED_V5_UUID_ATTRIBUTE

The `hash_based_v5_uuid_attribute` entity may be deprecated in the future. Please refer to Section 4.3.2.12 .

4.3.2.4 V4_UUID_ATTRIBUTE

The `v4_uuid_attribute` entity represents UUID identifier information. This entity collects the value of one of the UUID subtypes and the STEP entity that that UUID is assigned to. A `uuid_attribute` associates a UUID with an ordered collection of product data items. A `v4_uuid_attribute` is a `uuid_attribute` that provides a UUID that conforms to version

4 of the relevant rfc. Version 4 UUIDs are generated randomly and cannot be relied on between iterations of the product.

Note that there is an error in v4_uuid_attribute.wr1 related to character position of the version attribute. That will be corrected in ed5. The where rule shall be ignored during instance data validation.

4.3.2.5 UUID_ATTRIBUTE_WITH_APPROXIMATE_LOCATION

The `uuid_attribute_with_approximate_location` is a subtype of `uuid_attribute` that provides an approximate location in cartesian space of an item or items that has a UUID assigned.

4.3.2.6 UUID_RELATIONSHIP

The `uuid_relationship` relates two UUIDs and provides a role for that relationship.

4.3.2.6.1 UUID_RELATIONSHIP_ROLE

The `uuid_relationship_role` enumerates the permitted roles associated with a `uuid_relationship`. The allowed roles are SUPERSEDES, MERGE, SPLIT, DERIVE_FROM, SAME_AS, and SIMILAR_TO.

4.3.2.6.2 Example Usage of UUID_RELATIONSHIP for two different UUID_RELATIONSHIP_ROLEs

UUID Treatment for Surface Split

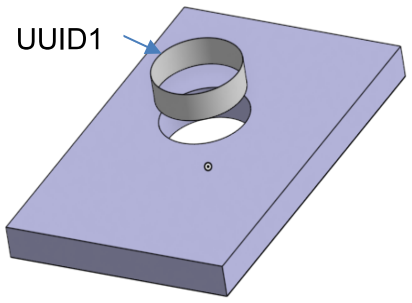


Figure 2a - Preprocessor - Single Cylinder

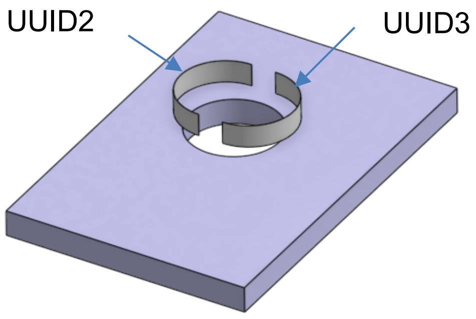


Figure 2b - Postprocessor - Half Cylinders

In Figure 2a and Figure 2b above, the native sending system models a hole feature as a single cylindrical surface and its STEP preprocessor assigns a UUID (UUID1) to the `advanced_face` for that surface on export. The receiving system models hole features as a pair of cylindrical surfaces and its STEP postprocessor assigns a single UUID (UUID2) to the set of two `advanced_face` entities created for the original surface received during import.

To maintain traceability between the original as-exported entity and the as-received entities, the post-processor ***must capture the relationship between UUIDs internally***. When reexporting is needed by the receiver, the system will use UUID_RELATIONSHIP as shown below to preserve that traceability on export.

Preprocessor…

```
#123 = ADVANCED_FACE(…);
#456 = V5_UUID_ATTRIBUTE(<UUID1>, UUID_SET_ITEM((#123)));
```

Postprocessor (when/if reexporting)…

```
#124 = ADVANCED_FACE(…);
#125 = ADVANCED_FACE(…);
#457 = V5_UUID_ATTRIBUTE(<UUID2>, UUID_SET_ITEM((#124,#125)));
#888 = UUID_RELATIONSHIP(<UUID3>,<UUID1>,<UUID2>, .SPLIT.);
```

UUID Treatment for Surface Merge

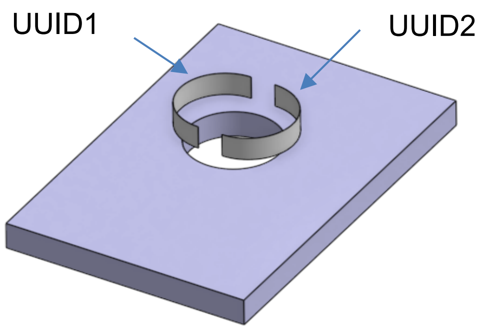


Figure 3a - Preprocessor - Half Cylinders

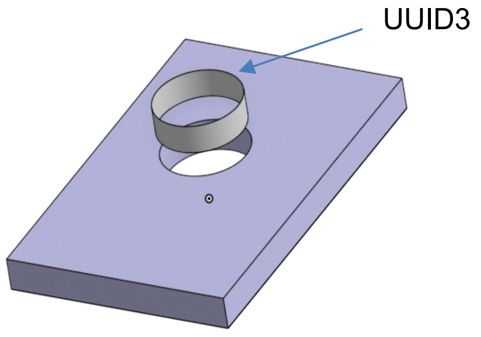


Figure 3b - Postprocessor - Single Cylinder

In Figure 3a and Figure 3b above, the native sending system models a hole feature as a pair of half cylindrical surfaces and its STEP preprocessor assign a UUID (UUID1) to the list of `advanced_face` entities for those surfaces on export. The receiving system models hole features as a single cylindrical surface and its STEP postprocessor assigns a UUID (UUID2) to that `advanced_face` entity during import while including a `uuid_relationship` that captures the fact that the new `advanced_face` (UUID2) is merged from the original `advanced face` list (UUID1) [see below].

Preprocessor…

```
#123 = ADVANCED_FACE(…);
#124 = ADVANCED_FACE(…);
#456 = V5_UUID_ATTRIBUTE(<UUID1>, UUID_SET_ITEM((#123,#124)));
```

Postprocessor (when/if reexporting)…

```
#125 = ADVANCED_FACE(…);
#458 = V5_UUID_ATTRIBUTE(<UUID2>, UUID_SET_ITEM((#125)));
#888 = UUID_RELATIONSHIP(<UUID3>,<UUID1>,<UUID2>, .MERGE.);
```

Example 3 – Example of use of UUID_LIST _ITEM to represent an ordered list of entities along a direction vector from top surface to bottom surface

Preprocessor…

```
#123 = ADVANCED_FACE(…); entry surface
#124 = ADVANCED_FACE(…);  inclined cyl half
#125 = ADVANCED_FACE(…);  second inclined cyl half
#126 = ADVANCED_FACE(…);  exit surface
#150 = EDGE_CURVE(…);   entry inclined cyl edge half
#151 = EDGE_CURVE(…);   entry second inclined cyl edge half
#153 = EDGE_CURVE(…);   exit inclined cyl edge half
#154 = EDGE_CURVE(…);   exit second inclined cyl edge half

#456 = V5_UUID_ATTRIBUTE(<UUID1>, UUID_LIST_ITEM
           (((#123),(#150,#151),(#124,#125),(#153,#154),(#126))));
```

Postprocessor (when/if reexporting)…

```
#123 = ADVANCED_FACE(…); entry surface
#124 = ADVANCED_FACE(…);  inclined cyl half
#125 = ADVANCED_FACE(…);  second inclined cyl half
#126 = ADVANCED_FACE(…);  exit surface
#150 = EDGE_CURVE(…);   entry inclined cyl edge half
#151 = EDGE_CURVE(…);   entry second inclined cyl edge half
#153 = EDGE_CURVE(…);   exit inclined cyl edge half
#154 = EDGE_CURVE(…);   exit second inclined cyl edge half

#456 = V5_UUID_ATTRIBUTE(<UUID1>, UUID_LIST_ITEM
           (((#123),(#150,#151),(#124,#125),(#153,#154),(#126))));
```

4.3.2.7 UUID_PROVENANCE

The `uuid_provenance` is the specification of a sequence of `uuid_relationships` that provides a historical record of those relationships. The sequence is a simple list form.

4.3.2.8 UUID_CONTEXT_ROLE

The `uuid_context_role` associates a role (non-empty string value) to a UUID. Preprocessors are recommended to populate this string value as “design_iteration” or “downstream_manufacturing”, as appropriate.

4.3.2.9 uuid_tree_node

A uuid_tree_node is one of (uuid_leaf_node, uuid_internal_node, uuid_root_node). There are two optional attributes node_1 and node_2 that specify lower level nodes. A uuid_tree_node is ABSTRACT and shall not be populated by itself.

4.3.2.10 uuid_leaf_node

A uuid_leaf_node is a subtype of uuid_tree_node that is a leaf node in a tree. A uuid_leaf_node specifies a uuid_attribute as its data attribute. A uuid_leaf_node shall be referenced by two uuid_tree_nodes (but not itself or another uuid_leaf_node).

4.3.2.11 uuid_internal_node

A uuid_internal_node is a subtype of uuid_tree_node that is internal to the tree. It may be referenced by one uuid node (that is not a uuid_leaf_node) and shall reference two lower level nodes.

Note: For initial trial the merkle tree is out of scope. No tree entities need be populated.

4.3.2.12 uuid_root_node

A uuid_root_node is a subtype of uuid_tree_node that is the root node in a tree. It shall not be referenced by other nodes and shall reference two lower level nodes. The uuid_root_node provides the hash function for the tree.

4.3.3 RECOMMENDATIONS for UUIDs

Preprocessor and Postprocessor Recommendations: All preprocessor and postprocessor applications ***shall*** be able to store and retrieve UUIDs and their entity assignments when needed. Applications ***shall*** be able to display to the user, on command, or via API-based queries for automated consumption/verification, any or all assigned UUIDs and any or all reference UUIDs and the entities to which they belong, using appropriate filters. The method of user display – model tree, parameter table, analysis table, or exported file and the content of the entity information displayed – is left to the vendor to determine.

A postprocessor ***shall*** determine whether a received UUID is owned by them or by some other system or entity by computing a UUID for the entity in question using the method of Section 4.1 above and comparing the as-received UUID to the as-computed UUID. If the UUIDs match, then the model object is owned by them. If, on the other hand, the UUIDs do not match, then the model object is owned by an external system or entity and the UUID and the object to which it refers is a reference UUID and object and the receiving system ***shall not*** modify it.

A preprocessor may also contain data previously imported from an external source and, therefore, ***shall not*** overwrite preexisting UUIDs or the data assigned to those UUIDs when exporting.

Preprocessor Recommendations: All preprocessors must generate UUIDs for each entity that they wish to persistently identify.

Each Product entity and Product Version entity ***shall*** have a UUID assigned. Each Semantic PMI entity -- dimensions, tolerances, datum tags and targets, surface finishes, and model notes -- ***shall*** have a UUID assigned. The following topological entities – faces, edges, as well as supplemental geometry entities that are used as reference for the above Semantic PMI entities in the pre-processing system – ***shall*** also have UUIDs assigned. Topological vertices ***shall not*** be assigned UUIDs.

In addition, User Defined Attributes (UDAs) may be assigned to product, geometry, or PMI in support of the downstream use case and these UDAs ***shall*** also have UUIDs assigned.

Preprocessors ***shall*** ensure that all UUIDs assigned to entities from the CAD model as described above must be maintained and be stable from one iteration of the CAD model to the next, i.e. an entity will retain the same UUID from model iteration to iteration, from CAD session to session, or from a session on machine X to a session on a different machine Y as long as the entity exists in the data. This rule applies at all levels, i.e., to Product and product version as well as to annotation (PMI) entities, and to associated geometry and topology or supplemental geometry entities. When new entities are created by the CAD user, new UUIDs ***shall*** be assigned to those new entities. When entities are deleted from the model, their UUIDs ***must not be reused***.

It is ***required*** that pre-processors, that own the original CAD data, will publish UUIDs as described in Section 4.1 above as identifiers within the Data Section of the STEP file using the structures described in Section 4.2.2 above. Though pre-processors are allowed to use the Anchor Section method as an alternate approach, this method does not support iterative exchange or downstream use and ***shall not*** to be used.

Also, preprocessors shall not populate `id_attribute` nor `aggregate_id_attribute` to assign a UUID to data.

Postprocessor Recommendations: Postprocessors ***shall*** support the reading of UUIDs when those UUIDs are published within the Data Section. UUIDs published within the Anchor Section ***shall not*** be used. Postprocessors ***shall*** retain incoming UUIDs for all identified geometry and topological entities, supplemental geometry entities, PMI entities, UDA entities and product and product version entities read.

Postprocessors ***shall*** ignore `id_attribute` and `aggregate_id_attribute` assigning UUID to data.

Related Entities:

N/A

4.3.4 RECOMMENDATIONS for DESIGN ITERATION

Preprocessor Recommendations:

All preprocessors ***shall*** generate UUIDs for each entity that they wish to permanently identify and must ensure that all UUIDs assigned to entities from the CAD model as described above must be maintained and be stable from one iteration of the CAD model to the next. (Ref – Sections 4.2.2 and 4.2.3 above and Section 4.3.5 below).

All preprocessors ***shall also export product and product version information*** with identifying UUIDs as specified Section 4.2.1 above.

Postprocessor Recommendations:

Initial Import -

All postprocessors, upon first import of a STEP model, ***shall*** retain incoming UUIDs for all identified STEP entities and map UUIDs assigned to the equivalent internal CAD entities as imported. How this mapping is handled in each postprocessing application is left to the individual application to manage but a table of mappings should be retained for later use in subsequent iteration.

All postprocessors, upon the first import of the STEP model, ***shall*** retain in the mapping the version information specified in Section 4.2.1 above as well as the product UUID and STEP file name information for the model for later use in subsequent iteration.

Subsequent to postprocessing, the imported model can be used for further design including the addition of new entities (geometry, features, datums, PMI, UDAs, process information, etc) ***as long as none of the imported entities that had UUIDs assigned are modified in any way.***

The postprocessing system ***shall***, if new content is added, create and assign UUIDs to new entities as needed to further facilitate iterative exchange using the method described above for preprocessors.

Subsequent Import -

Subsequent to the initial import above, a postprocessor ***shall***, when loading a new STEP file for which an existing imported model exists in memory (i.e., that can be identified as having matching product UUIDs and file name information), replace all existing imported entities with the newly imported content, but will ensure that any native entities previously added will have their references preserved (reassigned to the same imported entities). This process will allow automated update of child references, thus preserving design intent from iteration to iteration.

Related Entities:

N/A

4.3.5 RECOMMENDATIONS FOR DOWNSTREAM CONSUMPTION

Systems exchanging data for downstream use need to include mechanisms for collecting geometric “features” that represent elements of product shape that typically represent manufacturing and / or inspection operations. These geometric “features” need to be tracked and therefore require that UUIDs be assigned as appropriate.

Note that, for downstream consumption purposes, the ‘design’ context should be replaced by a ‘manufacturing’ context.

4.3.5.1 System-specific treatment of PIDs for split Holes (for R57J and beyond)

For test rounds including R57J and beyond, preprocessors shall assign UUIDs for split entities for holes – `advanced_face` (cylinders, bspline_surfaces, and conical surfaces) and `edge_curve` (circular curves, elliptical curves [for non-normal hole penetrations], and bspline curves) according to the following system-specific topology models below to present consistent identification of logical hole elements across systems.

In doing so, all three system kernel types end up with 3 UUID for logical holes:

- NX – one 360-degree cylinder `advanced_face` (whether cylindrical or bspline) with one internal ID and 2 (upper and lower) 360-degree `edge_curve`s (whether circular or bspline or elliptical) with an internal ID for each of them resulting in 3 UUIDs being created for the logical hole.

```
#209=CIRCLE('',#208,5.0);
#210=EDGE_CURVE('no 66',#204,#204,#209,.T.);
#232=CIRCLE('',#231,5.0);
#233=EDGE_CURVE('no 67',#227,#227,#232,.T.);

#259=CYLINDRICAL_SURFACE('',#258,5.0);
#260=ADVANCED_FACE('no 57',(#251,#254),#259,.F.);

#278=V5_UUID_ATTRIBUTE('D6DEF23B-76ED-A5DE-03CC-
78238E586F87',UUID_SET_ITEM((#233)));
#281=V5_UUID_ATTRIBUTE('F8DED3D2-836B-826A-1BA4-
A143B898F1FC',UUID_SET_ITEM((#260)));
#288=V5_UUID_ATTRIBUTE('42A3737E-E39D-7F49-9D63-
5EF0B0E48B98',UUID_SET_ITEM((#210)));
```

- CATIA (V5 and 3DX) – 2 paired 180-degree half-cylinder `advanced_face`s (whether cylindrical or bspline) with only one internal ID and 2 (upper and lower) paired 180-degree half- `edge_curve`s (whether circular or bspline or elliptical) with an internal IDs for each of them for 3 UUIDs for the logical hole.

```
#75=CIRCLE('',#74,5.0);
#76=EDGE_CURVE('no 18280',#68,#70,#75,.T.);
#92=CIRCLE('',#91,5.0);
#93=EDGE_CURVE('no 18271',#79,#87,#92,.T.);

#107=CYLINDRICAL_SURFACE('',#106,5.0);
#108=ADVANCED_FACE('no 18294',(#102),#107,.F.);

#147=CIRCLE('',#146,5.0);
#148=EDGE_CURVE('no 18263',#70,#68,#147,.T.);
#165=CIRCLE('',#164,5.0);
#166=EDGE_CURVE('no 18254',#87,#79,#165,.T.);

#175=CYLINDRICAL_SURFACE('',#174,5.0);
#176=ADVANCED_FACE('no 18300',(#170),#175,.F.);

#305=V5_UUID_ATTRIBUTE('889E512F-BB51-5E05-79BB-
91DC34103A72',UUID_SET_ITEM((#108,#176)));
#309=V5_UUID_ATTRIBUTE('B1B03397-9527-3E32-01CD-
6B2B0B884D9C',UUID_SET_ITEM((#76,#148)));
#318=V5_UUID_ATTRIBUTE('2723D061-C3BF-4F9E-98AA-
CFB73FEEAB98',UUID_SET_ITEM((#93,#166)));
```

- Creo – 2 paired 180-degree half-cylinder `advanced_face`s (whether cylindrical or bspline) with 2 internal IDs and 2 (upper and lower) paired 180-degree half-`edge_curve`s (whether circular or bspline or elliptical) with an internal ID for each pair of them for 3 UUIDs for the logical hole. For Creo, preprocessors will concatenate the 2 internal IDs for a pair into a single Namestring when generating the UUID and assigning the `advanced_face`s or `edge`s as `UUID_SET_ITEM` pairs.

```
#69=B_SPLINE_CURVE_WITH_KNOTS('',3,(…),.UNSPECIFIED.,.F.,.U.,
(…),(…),.UNSPECIFIED.);
#70=EDGE_CURVE('no 82',#52,#54,#69,.T.);
#86=B_SPLINE_CURVE_WITH_KNOTS('',3,(…),.UNSPECIFIED.,.F.,.U.,
(…),(…),.UNSPECIFIED.);
#87=EDGE_CURVE('no 81',#54,#52,#86,.T.);
#217=B_SPLINE_CURVE_WITH_KNOTS('',3,(…),.UNSPECIFIED.,.F.,.U.,
(…),(…),.UNSPECIFIED.);
#218=EDGE_CURVE('no 84',#202,#194,#217,.T.);
#250=B_SPLINE_CURVE_WITH_KNOTS('',3,(…),.UNSPECIFIED.,.F.,.U.,
(…),(…),.UNSPECIFIED.);
#251=EDGE_CURVE('no 83',#194,#202,#250,.T.);

#233=CYLINDRICAL_SURFACE('',#232,5.0);
#234=ADVANCED_FACE('no 77',(#228),#233,.F.);
#261=CYLINDRICAL_SURFACE('',#260,5.0);
#262=ADVANCED_FACE('no 75',(#256),#261,.F.);

#284=V5_UUID_ATTRIBUTE('8F2708AF-3DC1-FE6B-FD92-
0374C5528F7F',UUID_SET_ITEM((#218,#251)));
#291=V5_UUID_ATTRIBUTE('C581FFB4-78FC-7271-4CF7-
AD9DCDF7A907',UUID_SET_ITEM((#70,#87)));
#298=V5_UUID_ATTRIBUTE('2CF2DBA1-204F-E9A3-35D1-
A561B3D9EA4A',UUID_SET_ITEM((#234,#262)));
```

- Note – Since seam edges for holes (aligned with the axis of the cylinder) are not required for the formation of a logical hole, they have individual internal IDs and corresponding UUIDs without the requirement to pair them via a `UUID_SET_ITEM` regardless of which of the above system kernel types are being used.

4.3.5.2 UUID support for SURFACE_TEXTURE

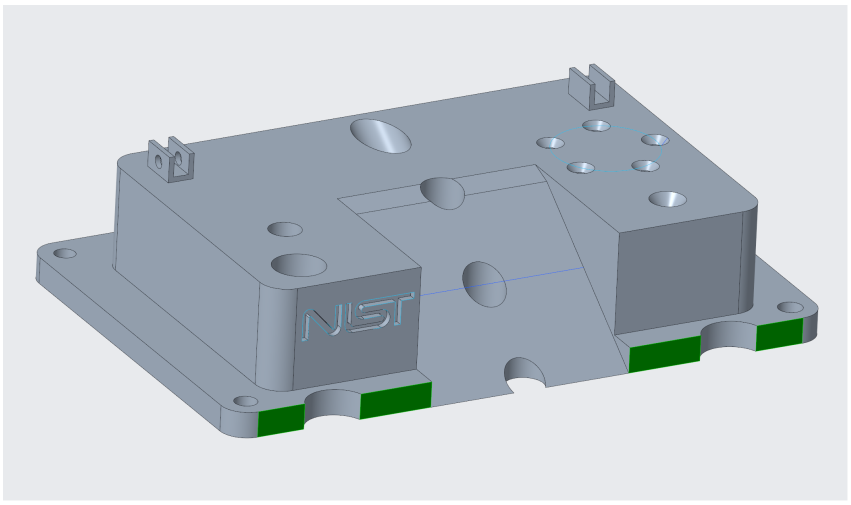
Vendors have reported that they cannot retrieve an internal persistent ID on individual surface texture parameters. The parameter information is embedded within a single surface texture structure. As such, the approach to use is to introduce UUID_LIST_ITEM. The first item in the list would be the surface texture and the remaining items in the list would be the surface texture parameters (see an example below).

```
#10=SURFACE_TEXTURE(...);

#20=SURFACE_TEXTURE_PARAMETER(...);

#30=SURFACE_TEXTURE_PARAMETER(...);

#100=UUID_ATTRIBUTE(<uuid>,UUID_LIST_ITEM(((#10), (#20,...,
(#30))));
```

4.3.5.3 System-specific treatment of PIDs for split planar face entities (for R57J and beyond)

Disjoint planar faces with one single internal ID can be created in Creo[4] (see below).

Figure 4 - Disjoint Planar Faces with one Internal ID (Creo)



To persistently identify these faces properly, the advanced_faces need to be collected within a UUID_SET_ITEM when assigning a UUID to the set. This collection is shown below.

```
#4 = ADVANCED_FACE('',(#11),#12,.T.);
#5 = ADVANCED_FACE('',(#13),#14,.T.);
#6 = ADVANCED_FACE('',(#15),#16,.T.);
#7 = ADVANCED_FACE('',(#17),#18,.T.);
#457 = V5_UUID_ATTRIBUTE(<uuid>, UUID_SET_ITEM((#4, #5, #6, #7)));
```

4.3.5.4 Geometric "Feature" Collection with UUIDs (for R58J and beyond only) -

For test rounds starting with R58J and beyond new PID test cases will introduce geometric feature collection as part of Bill of Operational Features (BOF) processing. A new Recommended Practice will be developed for R58J in support of BOF constructs.

For these BOFs, preprocessing systems ***shall*** create geometric feature groups for a particular class of design features, for example, for individual hole features (simple or complex) or for pattern features containing those hole features. A STEP instance diagram and P21 snippet are provided to capture an example of such a collection of entities and assignment of UUIDs to those collections. The example instance diagram and Part21 snippet are shown in Appendix B for a complex blind hole.

[4] Whether this split face concept exists in other systems (NX, V5, 3DX, etc) is under investigation.

Preprocessor Recommendations:

All preprocessors ***shall*** generate UUIDs for each entity that they wish to permanently identify and must ensure that all UUIDs assigned to entities from the CAD model as described above ***shall*** be maintained and be stable from one iteration of the CAD model to the next. (Ref – Sections 4.3.2 and 4.3.3 above).

All preprocessors ***shall also export product and product version information*** with identifying UUIDs as specified Section 4.3.1 above.

Postprocessor Recommendations:

All postprocessors, upon first import of a STEP model, ***shall*** retain incoming UUIDs for all identified STEP entities and map UUIDs assigned to the equivalent internal CAD entities as imported. How this mapping is handled in each postprocessing application is left to the individual application to manage but a table of mappings should be retained for later use either in subsequent iterations of design or for return of feedback from downstream processes.

All postprocessors, upon the first import of the STEP model, ***shall*** retain in the mapping the version information specified in Section 4.3.1 above as well as the product UUID and STEP file name information for the model for later use in subsequent iteration.

Subsequent to postprocessing, the imported model can be used for iteration or feedback including the addition of new entities (geometry, features, datums, PMI, UDAs, process information, etc) ***as long as none of the imported entities that had UUIDs assigned are modified in any way.***

The postprocessing system ***shall***, if new content is added, create and assign UUIDs to new entities as needed to further facilitate iterative exchange using the method described above for preprocessors.

Related Entities:

N/A

5 Express Diagrams

The EXPRESS entities and attributes used to support the complete requirements of entity identification for product, PMI, topology/geometry, supplemental geometry, and UDA are illustrated in the figures on the following pages. Note that all instances of terms in the diagrams shown in Figures 4 through 8 below having the characters “guid” in them are now replaced with the characters “uuid”.

Note that Figure 5 is rotated by 90° for better readability.

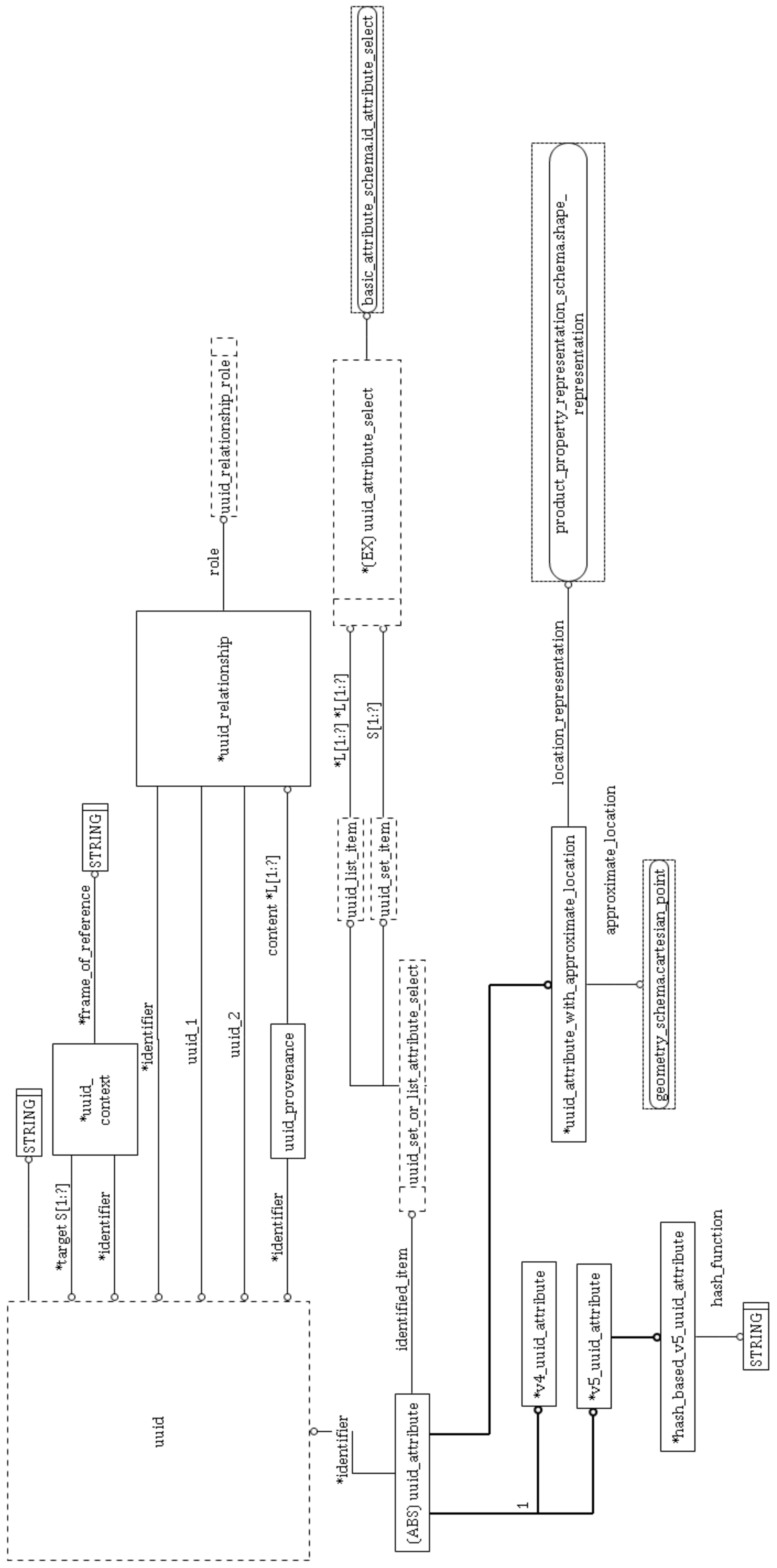


Figure 5: UUID Attribute Specific Schema Elements

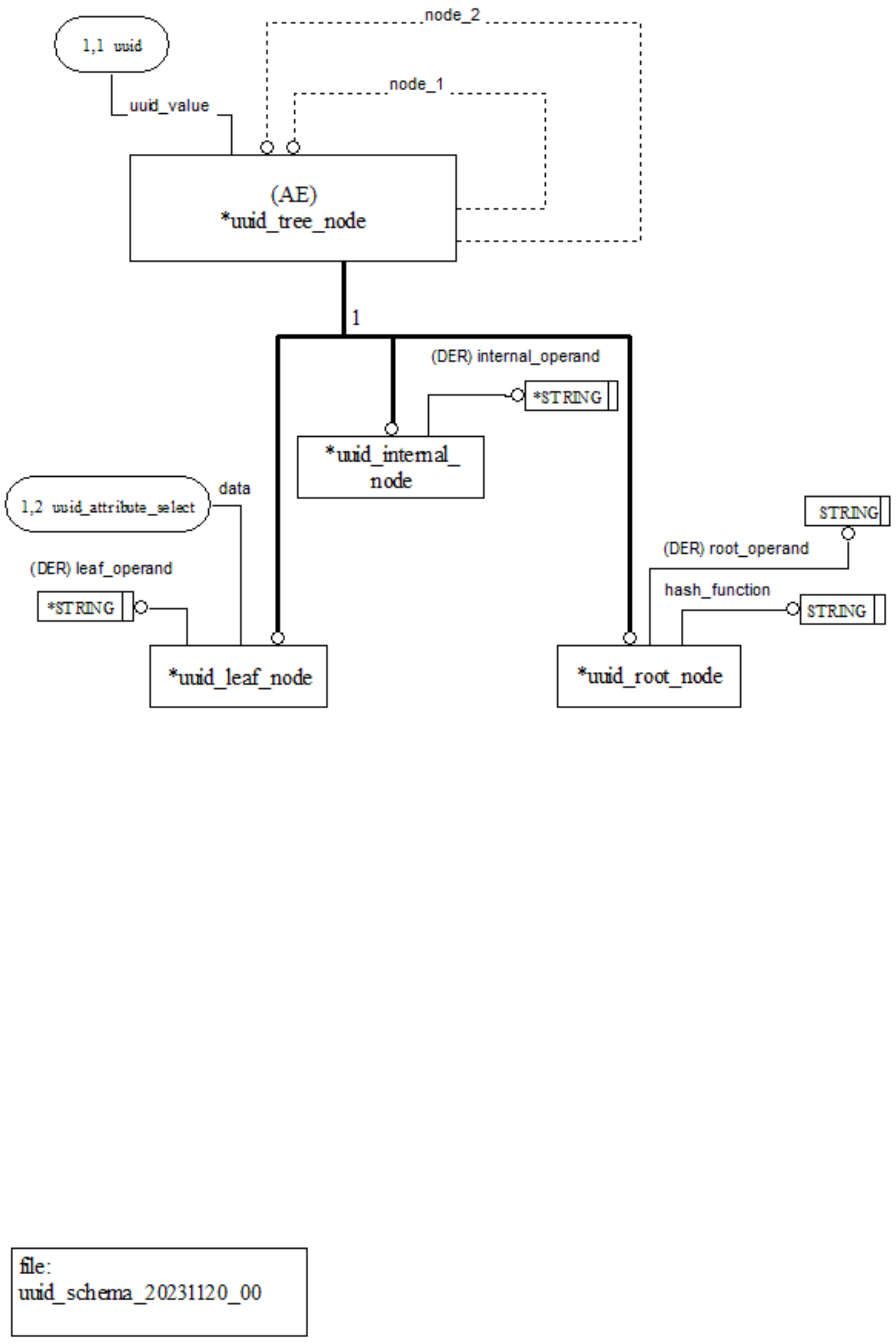


Figure 6: UUID Tree Specific Schema Elements

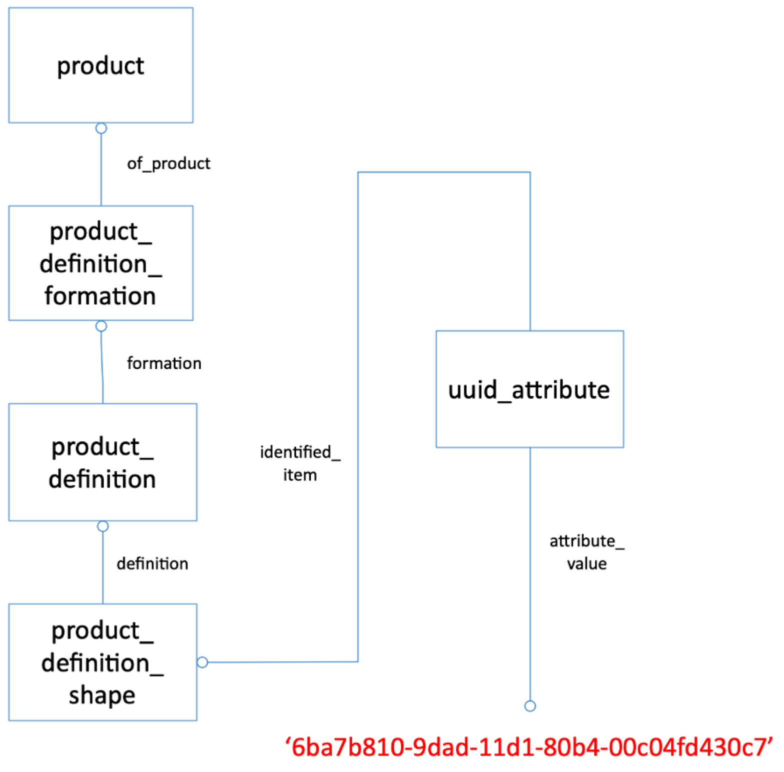


Figure 7: Entity Identifier for Product

(note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

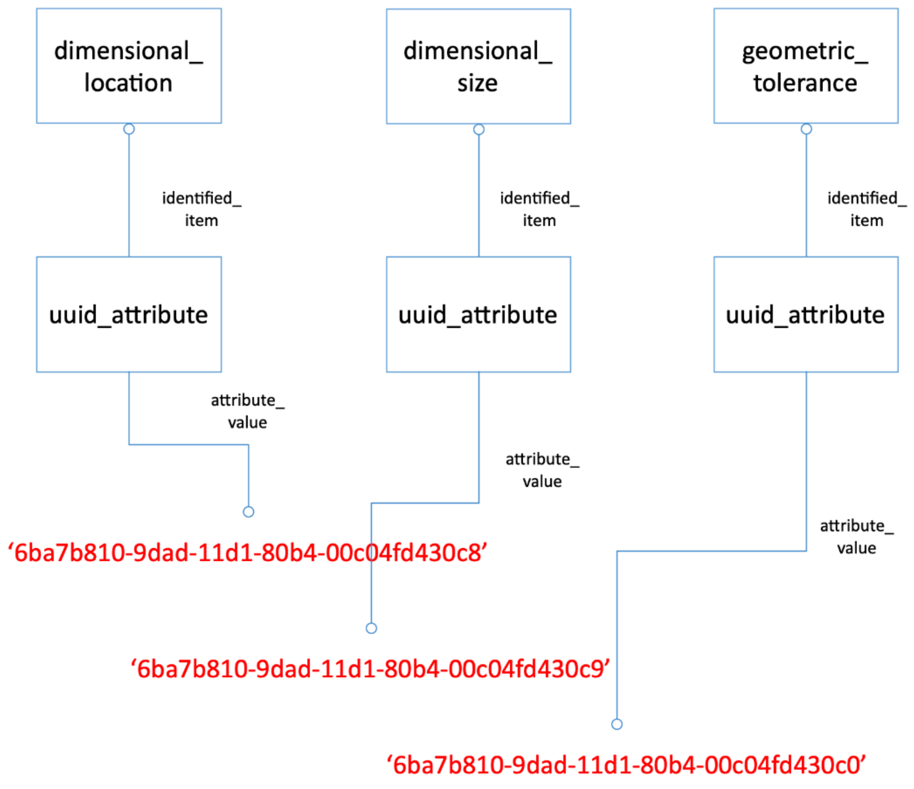


Figure 8: Entity Identifiers for PMI

(examples) (note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

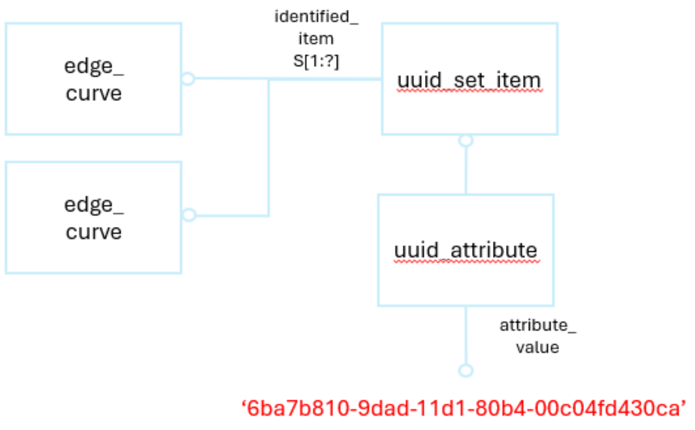


Figure 9a: Entity Identifier using uuid_set_item for Topology/Geometry

(note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

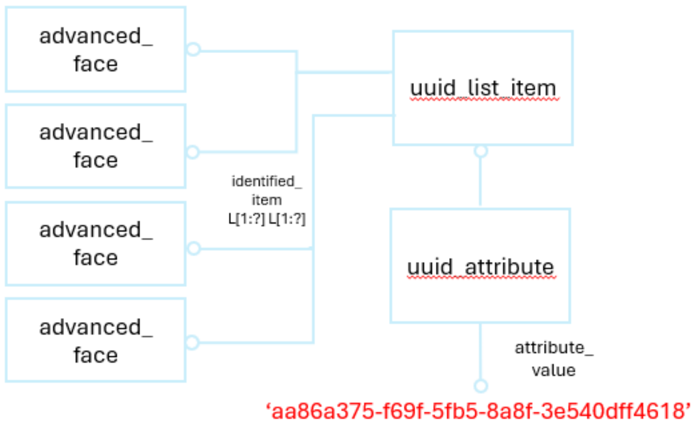


Figure 8b: Entity Identifier using uuid_list_item for Topology/Geometry

(note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

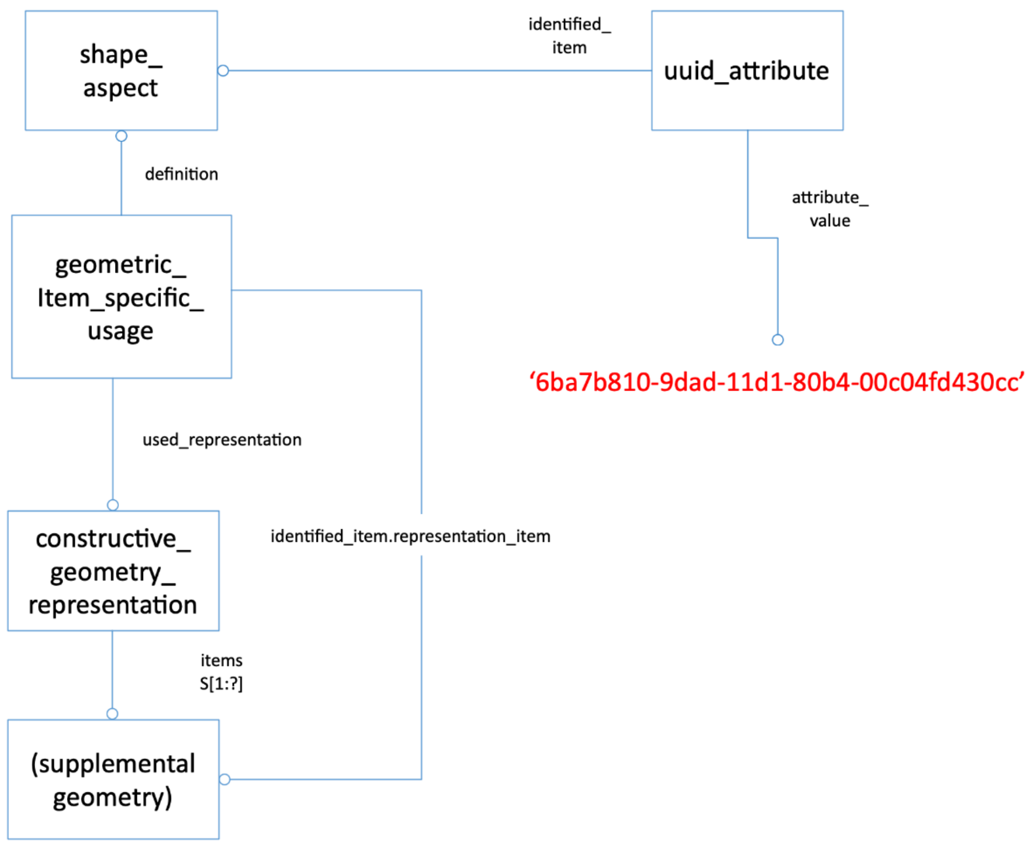

shape_ aspect
identified_ item
uuid_attribute
definition
attribute_ value
geometric_ Item_specific_ usage
'6ba7b810-9dad-11d1-80b4-00c04fd430cc'
used_representation
constructive_ geometry_ representation
identified_item.representation_item
items S[1:?]
(supplemental geometry)

Figure 10: Entity Identifier for Supplemental Geometry

(note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

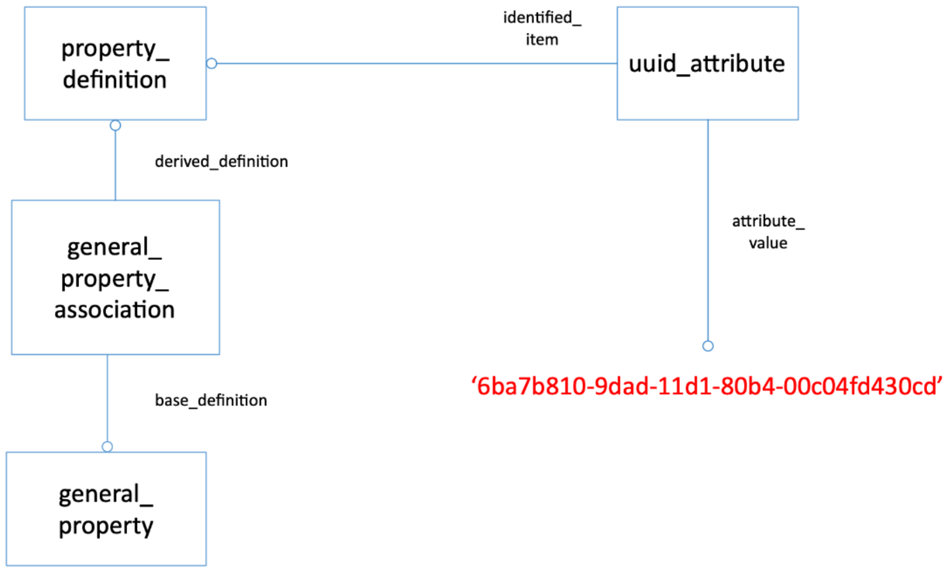


Figure 11: Entity Identifier for UDA

(note – uuid_attribute must be replaced by either v5_uuid_attribute or v4_uuid_attribute.)

Availability of Implementation Schemas

A.1 AP 242 Edition 4

The capabilities described in this Recommended Practices document require at least AP242 Edition 4 (2025) for implementation.

The longform EXPRESS schema for the fourth Edition of AP242 can be retrieved from:

https://standards.iso.org/iso/ts/10303/-442/ed-7/tech/express/mim_lf.exp

A.2 Complete list of Referenceable Entities by a UUID

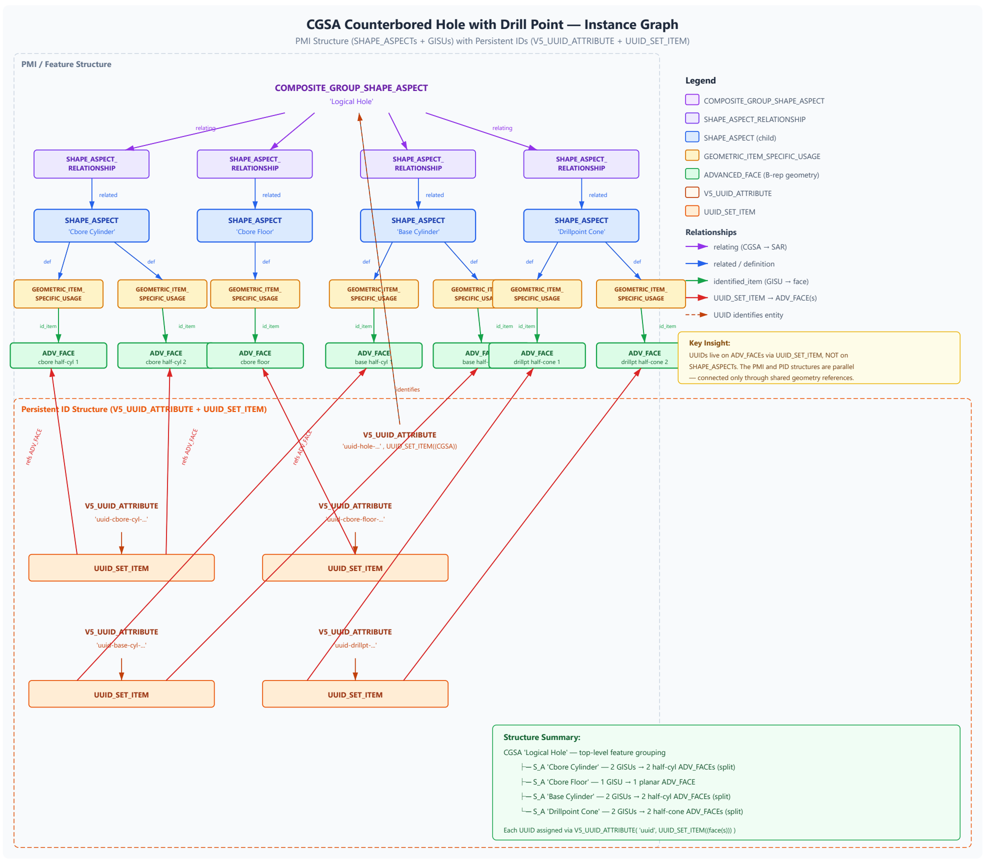
The complete list of entities that may be referenced by a UUID in AP 242 Edition 4 can be retrieved from (CAx-IF member access only):

https://nextcloud.boost-lab.net/nextcloud/index.php/f/179700

B. Bill of Operational Features (BOF) Example with UUID assignments

B.1 Instance diagram for Complex Hole (BOF)

Figure 12 - Instance diagram for Complex Hole (BOF)



Key Insight:
UUIDs live on ADV_FACEs via UUID_SET_ITEM, NOT on SHAPE_ASPECTs. The PMI and PID structures are parallel — connected only through shared geometry references.

Structure Summary:
CGSA 'Logical Hole' — top-level feature grouping
├— S_A 'Cbore Cylinder' — 2 GISUs → 2 half-cyl ADV_FACEs (split)
├— S_A 'Cbore Floor' — 1 GISU → 1 planar ADV_FACE
├— S_A 'Base Cylinder' — 2 GISUs → 2 half-cyl ADV_FACEs (split)
└— S_A 'Drillpoint Cone' — 2 GISUs → 2 half-cone ADV_FACEs (split)
Each UUID assigned via V5_UUID_ATTRIBUTE('uuid', UUID_SET_ITEM((face(s))))

B.2 Part 21 Snippet for Complex Hole (BOF)

```
/* ================================================================
   CGSA Counterbored Hole with Drill Point — Part 21 Fragment
   Reproduces the instance graph from CGSA_hole_instance_graph.svg
   Validated against SMRLv12 (N11651_SMRLv12_rc7)

   Structure:
     CGSA 'Logical Hole'
       +-- SA 'Cbore Cylinder'   -> 2 GISUs -> 2 ADV_FACEs (split)
       +-- SA 'Cbore Floor'      -> 1 GISU  -> 1 ADV_FACE
       +-- SA 'Base Cylinder'    -> 2 GISUs -> 2 ADV_FACEs (split)
       +-- SA 'Drillpoint Cone'  -> 2 GISUs -> 2 ADV_FACEs (split)

     Persistent IDs via V5_UUID_ATTRIBUTE with inline uuid_set_item
   ================================================================
*/

/* ---------- Context (assumed to exist in the full file) ----------
*/
#1 = PRODUCT('Hole_Part','Hole_Part','',(#2));
#2 = PRODUCT_CONTEXT('',#3,'mechanical');
#3 = APPLICATION_CONTEXT('core data for automotive design');
#4 = PRODUCT_DEFINITION('quality','',#5,#6);
#5 = PRODUCT_DEFINITION_FORMATION('','',#1);
#6 = PRODUCT_DEFINITION_CONTEXT('part definition',#3,'quality');
#7 = PRODUCT_DEFINITION_SHAPE('','',#4);

/* --- Shape representation (geometric model containing the faces) -
*/
#8 = SHAPE_REPRESENTATION('hole geometry',(#210,#211,#220,#230,#231,
       #240,#241),#9);
#9 = ( GEOMETRIC_REPRESENTATION_CONTEXT(3)
       GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT((#10))
       GLOBAL_UNIT_ASSIGNED_CONTEXT((#11,#12,#13))
       REPRESENTATION_CONTEXT('3D','3D context') );
#10 = UNCERTAINTY_MEASURE_WITH_UNIT(LENGTH_MEASURE(1.0E-06),#11,
        '','maximum tolerance');
#11 = ( LENGTH_UNIT() NAMED_UNIT(*) SI_UNIT(.MILLI.,.METRE.) );
```

```
#12 = ( NAMED_UNIT(*) PLANE_ANGLE_UNIT() SI_UNIT($,.RADIAN.) );
#13 = ( NAMED_UNIT(*) SI_UNIT($,.STERADIAN.) SOLID_ANGLE_UNIT() );


/* --- Link shape_representation to product_definition_shape -------
*/
#14 = SHAPE_DEFINITION_REPRESENTATION(#7,#8);


/* ================================================================
   PMI / FEATURE STRUCTURE
   ================================================================
*/


/*  --- Top-level: COMPOSITE_GROUP_SHAPE_ASPECT --------------------
*/
/*     schema: shape_aspect_definition_schema (ISO 10303-47 ed7)
       composite_group_shape_aspect            SUBTYPE            OF
(composite_shape_aspect)
       composite_shape_aspect SUBTYPE OF (shape_aspect)
         INVERSE component_relationships : SET[2:?] OF
           shape_aspect_relationship FOR relating_shape_aspect
       shape_aspect:       name,       description,       of_shape,
product_definitional */
#100 = COMPOSITE_GROUP_SHAPE_ASPECT('Logical Hole',
         'Top-level counterbored hole with drillpoint', #7, .T.);


/* --- Branch 1: Cbore Cylinder (split into 2 half-cylinders) -----
*/
#110 = SHAPE_ASPECT('Cbore Cylinder',
         'Counterbore cylindrical surface', #7, .T.);
#111 = SHAPE_ASPECT_RELATIONSHIP('','CGSA to cbore cylinder',
         #100, #110);


/*    GISU: definition=shape_aspect, used_representation=shape_model,
       identified_item=geometric_model_item (the ADV_FACE)          */
#112 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #110, #8, #210);
#113 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #110, #8, #211);


/* --- Branch 2: Cbore Floor (single planar face) -----------------
*/
#120 = SHAPE_ASPECT('Cbore Floor',
         'Counterbore bottom planar surface', #7, .T.);
```

```
#121 = SHAPE_ASPECT_RELATIONSHIP('','CGSA to cbore floor',
         #100, #120);

#122 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #120, #8, #220);

/* --- Branch 3: Base Cylinder (split into 2 half-cylinders) ------
*/
#130 = SHAPE_ASPECT('Base Cylinder',
         'Main hole cylindrical surface', #7, .T.);
#131 = SHAPE_ASPECT_RELATIONSHIP('','CGSA to base cylinder',
         #100, #130);

#132 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #130, #8, #230);
#133 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #130, #8, #231);

/* --- Branch 4: Drillpoint Cone (split into 2 half-cones) --------
*/
#140 = SHAPE_ASPECT('Drillpoint Cone',
         'Drill point conical surface', #7, .T.);
#141 = SHAPE_ASPECT_RELATIONSHIP('','CGSA to drillpoint cone',
         #100, #140);

#142 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #140, #8, #240);
#143 = GEOMETRIC_ITEM_SPECIFIC_USAGE('','', #140, #8, #241);


/* ================================================================
   B-REP GEOMETRY (Placeholder ADVANCED_FACEs)
   ================================================================
*/

/* --- Cbore Cylinder faces ----------------------------------------
*/
#210 = ADVANCED_FACE('cbore half-cyl 1', (), #215, .T.);
#211 = ADVANCED_FACE('cbore half-cyl 2', (), #215, .T.);
#215 = CYLINDRICAL_SURFACE('', #216, 8.0);
#216 = AXIS2_PLACEMENT_3D('cbore axis', #217, #218, #219);
#217 = CARTESIAN_POINT('', (0.0, 0.0, 0.0));
#218 = DIRECTION('', (0.0, 0.0, 1.0));
#219 = DIRECTION('', (1.0, 0.0, 0.0));
```

```
/* --- Cbore Floor face ---------------------------------------
*/
#220 = ADVANCED_FACE('cbore floor', (), #225, .T.);
#225 = PLANE('', #226);
#226 = AXIS2_PLACEMENT_3D('cbore floor plane', #227, #228, #229);
#227 = CARTESIAN_POINT('', (0.0, 0.0, -5.0));
#228 = DIRECTION('', (0.0, 0.0, 1.0));
#229 = DIRECTION('', (1.0, 0.0, 0.0));

/* --- Base Cylinder faces ------------------------------------
*/
#230 = ADVANCED_FACE('base half-cyl 1', (), #235, .T.);
#231 = ADVANCED_FACE('base half-cyl 2', (), #235, .T.);
#235 = CYLINDRICAL_SURFACE('', #236, 4.0);
#236 = AXIS2_PLACEMENT_3D('base axis', #237, #238, #239);
#237 = CARTESIAN_POINT('', (0.0, 0.0, -5.0));
#238 = DIRECTION('', (0.0, 0.0, 1.0));
#239 = DIRECTION('', (1.0, 0.0, 0.0));

/* --- Drillpoint Cone faces ----------------------------------
*/
#240 = ADVANCED_FACE('drillpt half-cone 1', (), #245, .T.);
#241 = ADVANCED_FACE('drillpt half-cone 2', (), #245, .T.);
#245 = CONICAL_SURFACE('', #246, 4.0, 1.0471975511966);
#246 = AXIS2_PLACEMENT_3D('drillpt axis', #247, #248, #249);
#247 = CARTESIAN_POINT('', (0.0, 0.0, -15.0));
#248 = DIRECTION('', (0.0, 0.0, 1.0));
#249 = DIRECTION('', (1.0, 0.0, 0.0));



/* ================================================================
   PERSISTENT ID STRUCTURE
   Schema: uuid_attribute_schema (ISO 10303-41 ed8)

   KEY POINT: uuid_set_item is a TYPE, not an ENTITY:
     TYPE uuid_set_item = SET [1:?] OF uuid_attribute_select;
   It appears inline as the identified_item value of uuid_attribute.
   v5_uuid_attribute SUBTYPE OF (uuid_attribute)
     identifier       : uuid  (STRING(36) FIXED)
     identified_item : uuid_set_or_list_attribute_select
       = SELECT(uuid_list_item, uuid_set_item)
   WHERE WR1: identifier[1] = '5'  (version digit)
   ================================================================
*/


/* --- UUID for CGSA itself ---------------------------------------
*/
#300 = V5_UUID_ATTRIBUTE('a1b2c3d4-e5f5-5a01-abcd-ef1234567890',
        UUID_SET_ITEM((#100)));


/* --- UUID for Cbore Cylinder (references 2 split faces) ---------
*/
#310 = V5_UUID_ATTRIBUTE('b2c3d4e5-f6a5-5b02-bcde-f12345678901',
        UUID_SET_ITEM((#210, #211)));


/* --- UUID for Cbore Floor (references 1 face) -------------------
*/
#320 = V5_UUID_ATTRIBUTE('c3d4e5f6-a7b5-5c03-cdef-123456789012',
        UUID_SET_ITEM((#220)));


/* --- UUID for Base Cylinder (references 2 split faces) ----------
*/
#330 = V5_UUID_ATTRIBUTE('d4e5f6a7-b8c5-5d04-defa-234567890123',
         UUID_SET_ITEM((#230, #231)));


/* --- UUID for Drillpoint Cone (references 2 split faces) --------
*/
#340 = V5_UUID_ATTRIBUTE('e5f6a7b8-c9d5-5e05-efab-345678901234',
         UUID_SET_ITEM((#240, #241)));


/* ================================================================
   END OF FRAGMENT
   ================================================================
*/
```