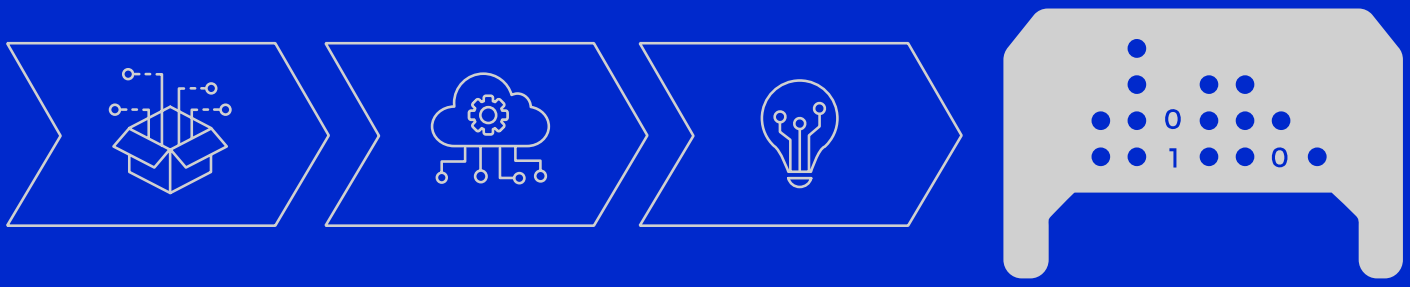

GUIDELINE

HOW TO CREATE A SUBMODEL TEMPLATE SPECIFICATION V1.1

June 2025

Imprint

Publisher
Industrial Digital Twin Association
Lyoner Strasse 18
60528 Frankfurt am Main
Germany
https://www.industrialdigitaltwin.org/

Version history

Date	Version	Changes made
2022-12-05	1.0	• Release of the Guideline
2025-06-01	1.1	• NEW: Additional workflows and rules for AsciiDoc • NEW: SMT Dropins • NEW: TemplateId and SemanticId • NEW: Marking arbitrary contents • Update to reflect V3.0 and V3.1 changes of the Specification of the AAS
2025-06-11	1.1	• Release of the Guideline

Contents

Figures

Tables

1 General

1.1 Overview

This document is an enabler for the IDTA specification series of Submodel template specifications. Each part of the mentioned series specifies the contents of a Submodel template (SMT) for the Asset Administration Shell (AAS). The Asset Administration Shell is described in [1], [2], [3] and [6]. First exemplary Submodel contents were described in [4] and [8], while the actual format of this document was derived by the "Administration Shell in Practice" [5].

The IEC working group IEC TC65 WG24 publishes the AAS as an international standard [7]. Figure 1 illustrates the AAS and Submodel template guiding the creation of Submodels. SubmodelElements and Submodel template elements may reference entries in concept repositories. The creators of Submodel template specification are designated as the Asset Administration Shell responsibles.

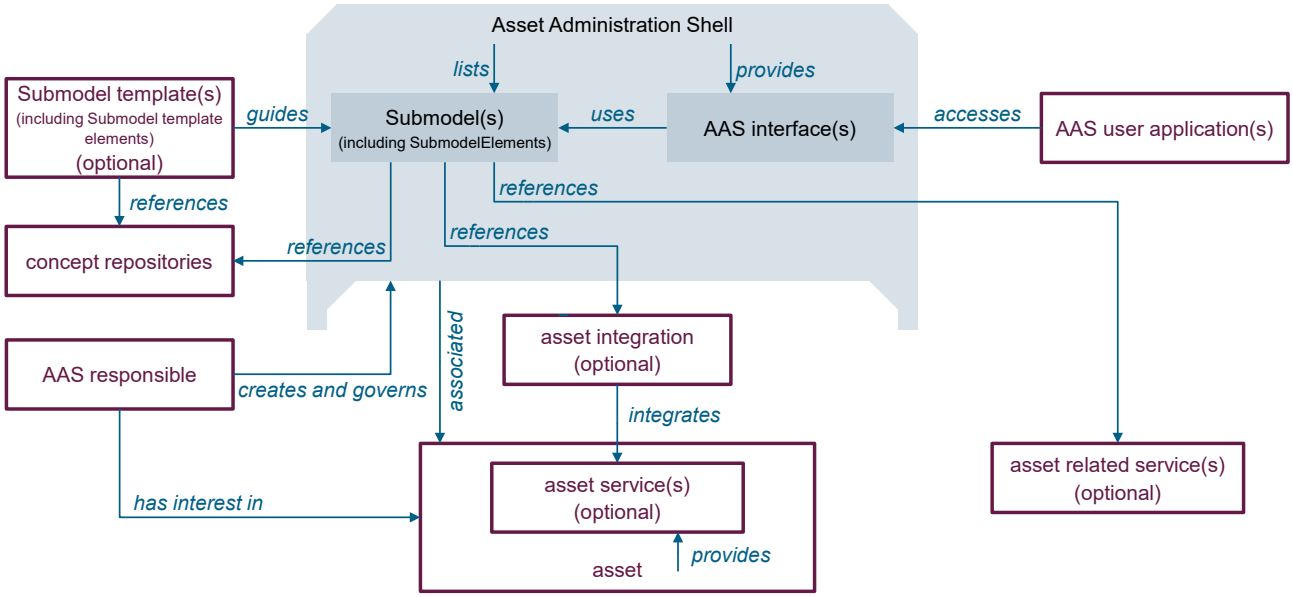

Figure 1 – Detailed overview of Asset Administration Shell and related roles [7]

Figure 2 illustrates two the Asset Administration Shell user applications, to use the interoperable Submodel information.

From the viewpoint of the meta model [6], a Submodel template is a Submodel with kind = Template.

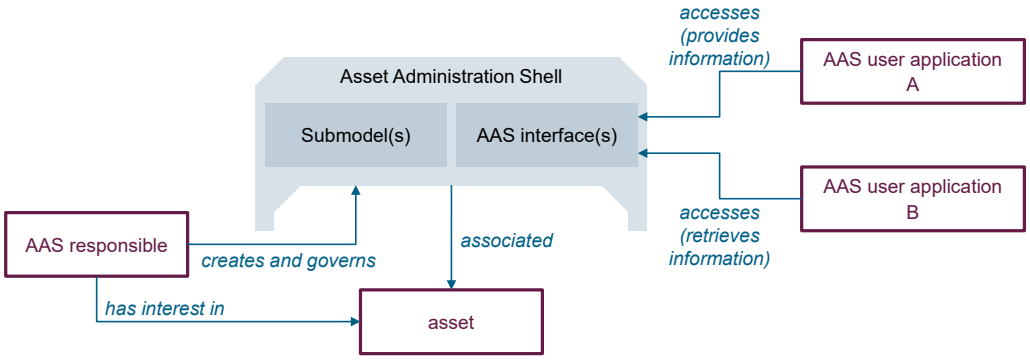

Figure 2 – Information exchange between AAS user applications [7]

This document facilitates the creation of Submodel template for the Asset Administration Shell. These Submodel templates will allow the Asset Administration Shell responsibles to create and provide standardized models, which represent certain aspects of an asset. Specifically, this document describes how Submodel template specifications for Submodel templates shall be created in order to maintain a set of common features and structures. For this purpose, different workflows are described.

The Submodel template comprises machine-readable information, such as AASX package files, but also human-readable information. The latter can be described in the form of a Submodel template specification.

Note: With the advent of AsciiDoc and Submodel template repositories, human-readable and machine-readable information will be joined together to a unified body of knowledge.

1.2 Scope of this document

The scope of this document is to provide a uniform human readable and machine-readable way of a Submodel template. It describes the possible workflows and necessary working steps to create a Submodel template. It allows multiple Submodel template teams to work in parallel on different Submodel templates, while maintaining a common structure and required features. Different human stakeholders shall be able to understand information and services associated with the particular Submodel template and realized Submodel instances.

Not scope of this document is the IDTA process for registering Submodel template; this is described by [R1].

1.3 Stakeholders

Stakeholders for reading and applying this document are, among others:

- members of a Submodel template working team creating a Submodel template specification
- developers creating technical provisions dealing with specific Submodels and their subject matter
- subject matter experts and users required to 'fill out' Submodels
- architects of the Asset Administration Shell and working groups

1.4 Abbreviations

The following abbreviations are used in this document:

Table 1 – Used abbreviations

Abbreviation	Description
AAS	Asset Administration Shell
AASX	AASX file package
DOC	Word processor document
SM	Submodel
SME	SubmodelElement
SMC	SubmodelElementCollection
SML	SubmodelElementList
SMT	Submodel template
WD	Working draft

XMI	XML Metadata Interchange

1.5 Conventions

For some terms, a special notation is used, in order to distinguish them from English text. Defined in [6], these include:

- Asset Administration Shell (AAS)
- Identifiable and Referable
- Qualifier
- Submodel
- SubmodelElement

Bibliography items are designated as references such as [1], resources are designated as reference such as [R1].

1.6 Meta model version

This document currently targets meta model version V3.0. Meta model version V3.1 is upcoming, and the contents described in this document will hold true, as well. Submodels (SM), SubmodelElementCollections (SMC) and SubmodelElementLists (SML) are used frequently as means of hierarchical structuring SubmodelElements.

1.7 Further general information

No further general information is given, yet.

2 Format of a SMT document

2.1 General

The creation of a SMT specification has multiple deliveries (see 3.1), including the human-readable part of SMT, currently a word processor document. This human-readable part is called SMT document. This document describes also, how this document can be described by markup elements (AsciiDoc).

The deliverables shall include human and machine-readable AASX files and may also include samples and more (see Table 3).

2.2 Structure of the human-readable part of SMT

For the structure of the content for the human-readable part of SMT, the SMT document, a template is available at IDTA office.

- Cover page (see Figure 3) which allows a high degree of recognition of the SMT series, providing:
  - uniform cover page design
  - IDTA document number (see section 2.3)
  - title of the SMT, often just called "the Submodel for..."
  - semantic version information (natural numbers for version, revision) (see section 2.4)
- Imprint
- Version history, giving information to the public, allowing to understand the evolution of the SMT over multiple versions
- Tables of contents, figures, tables
- Tables and figures shall provide titles
- Uniform scheme for clauses and subclauses, typically (see Table 2):

Table 2 – Common (sub-)sections of the human-readable part of SMT

(Sub-) Section	Aim
1 General	Provide general overview.
1.1 About this document	Provide very brief introduction to AAS and associated documents. Only referring to other documents. Typically taken unchanged from the template.
1.2 Scope of the Submodel	Give precise but concise definition of the scope, clarifying industry segments, stakeholders, life cycle steps, associated assets and subject matter to be represented.
1.3 Relevant standards	The use of existing standards is encouraged.
2 Approach of the Submodel	Different subsections explaining background and approaches, which lead to the current design of the SMT. Often, a UML diagram of the SMT entities is given. Often, a preview of how the subject matter could be presented to the user, is given, e.g. a screenshot of a plugin for the AASX Package Explorer, or inside an authoring system.
2.1 Assets	Shortly describes which assets are targeted by the SMT. In particular, it distinguishes between type assets and instance assets.
2.2 Use cases	Provides a brief overview of use cases for the SMT, e.g. by providing a table with 2-4 use cases and brief explanations (e.g. 2 sentences).

2.3 Data providers, consumers	Provides a short explanation of which roles/ organizations/ technical elements are providers and consumers of the Submodel data concerned in the SMT.
3 Submodel and SubmodelElements	Detailed description of the different entities by means of the defined table format.
4 {Further normative}	Further sections with normative content.
Appendix A – Explanations on used table formats	To allow the understanding of the definitions without extensive referring to other documents, this most important information shall be given in every SMT.
Appendix B – Bibliography	If possible, refer to more extensive material instead of re-describing.
Appendix C – Change log	If not the initial version, outline changes from the last version to the current version on a per-item basis.
Closing section	Link to IDTA.

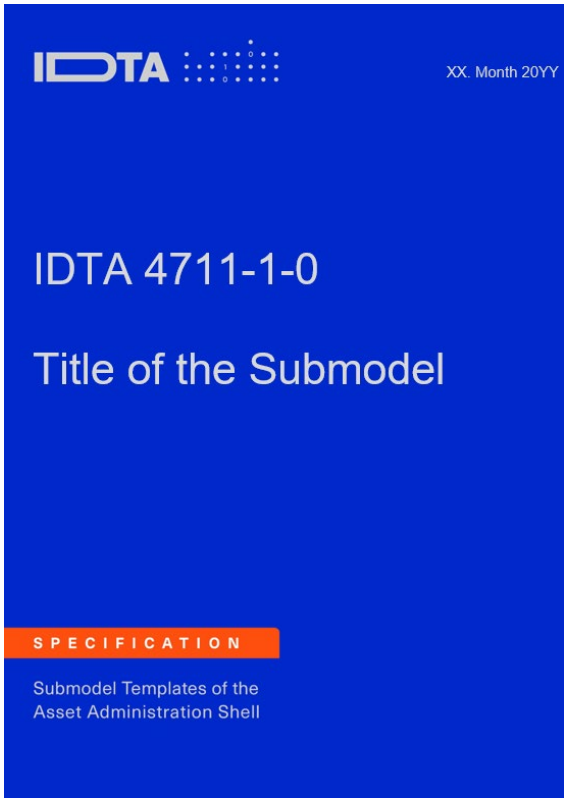

Figure 3 – Sample cover page of a SMT document

2.3 IDTA document number

The IDTA document number is retrieved from the IDTA office. The number for SMT starts at the 02000 number range (the 01000 - 01999 number range is reserved for meta models). It is possible to apply to the IDTA office to reserve a range of document numbers, e.g. for future extensions of the subject matter described.

2.4 Semantic version information of the SMT

The administrative information of Identifiables in the AAS entities comprises of two attributes: version and revision [6]. The version and revision are also indicated by the IDTA document number (e.g. IDTA 4711-1-0 stands für the SMT IDTA 4711 version 1 revision 0). For the SMT, this is seen as semantic versioning:

- The version designates the major version of the SMT. An increment typically indicates a breaking change, requiring an adoption of the information by a human.
- The revision designates a minor version of the SMT. An increment typically designates recognizable changes, fixes and feature improvements, which are not breaking changes.
  A revision requires a version. This means, if there is no version there is no revision either.
- If required, a third digit (e.g. IDTA 4711-1-1-4) might be introduced to differentiate publication of bug fixes. This digit is not reflected in version/ revision of administrative information.

3 Workflows

3.1 General

In this section, multiple possible workflows are described. Any workflow might be executed by an architect, however, reviews within the SMT team and with possible domain experts are considered essential. The workflow and the achieved results shall comply with the IDTA Process Description [R1].

3.2 Artifacts to be delivered

For the "(6) Review" phase of the process [R1] and for publication by IDTA, the following artifacts are mandatory/ optional (see Table 3).

Table 3 – Artifacts of a SMT Specification

Artifact	Description	Cardinality
SMT specification	Word processor document, representing the SMT specification at a whole	One
SMT model	Logical model with Submodel of kind = Template and respective ConceptDescriptions representing the template definition of the SMT. The SMT model might be represented by files (see below) or within a repository.	(implicit, see below)
Pure SMT AASX file	AASX file for the SMT model, containing a Submodel and SubmodelElements of kind = Template. No Qualifiers according 9.2, 9.3 shall be attributed.	One
Qualified SMT AASX file	An AASX for the SMT model, but SubmodelElements are attributed with Qualifiers, such as described in 9.2, 9.3.	ZeroToOne
Example AASX file(s)	AASX files with exemplary AAS, Submodels and SubmodelElements of kind = Instance, demonstrating the purpose of the SMT	ZeroToMany
Generic forms preset(s)	Option file(s) for the AasxPluginGenericForms in JSON format to allow easy creation and filling out of SMTs	ZeroToMany
Source format file for given figures	If the SMT specification uses figures for illustrations, the source format files shall be given for later maintenance.	ZeroToMany

3.3 Recommendations

The following sections describe multiple workflows which can be utilized to create a SMT specification. Each of these workflows is allowed to be used. However, in order to maintain quality of the SMT specifications and to streamline the set of specifications for the future, the following recommendations are stated by the IDTA:

(1) The use of one of the AsciiDoc workflows (see 3.7, 3.8) is recommended.

(2) The use of the Semantic driven workflow (see 3.6) combined with one of the AsciiDoc workflows (see 3.7, 3.8) is recommended, if a semantic definition in a concept repository makes sense, e.g. no semantic definition is already existing and the concepts might be used in other use cases, e.g. business-to-business application.

3.4 Document driven workflow

The document driven workflow is rather simple and can be executed with limited invocation of advanced tools. It is suitable for rather simple Submodels. Figure 4 demonstrates the workflow.

Note: The use of this workflow is deprecated. It might be useful for very small SMT specifications or for initial proposition steps prior a formal SMT specification.

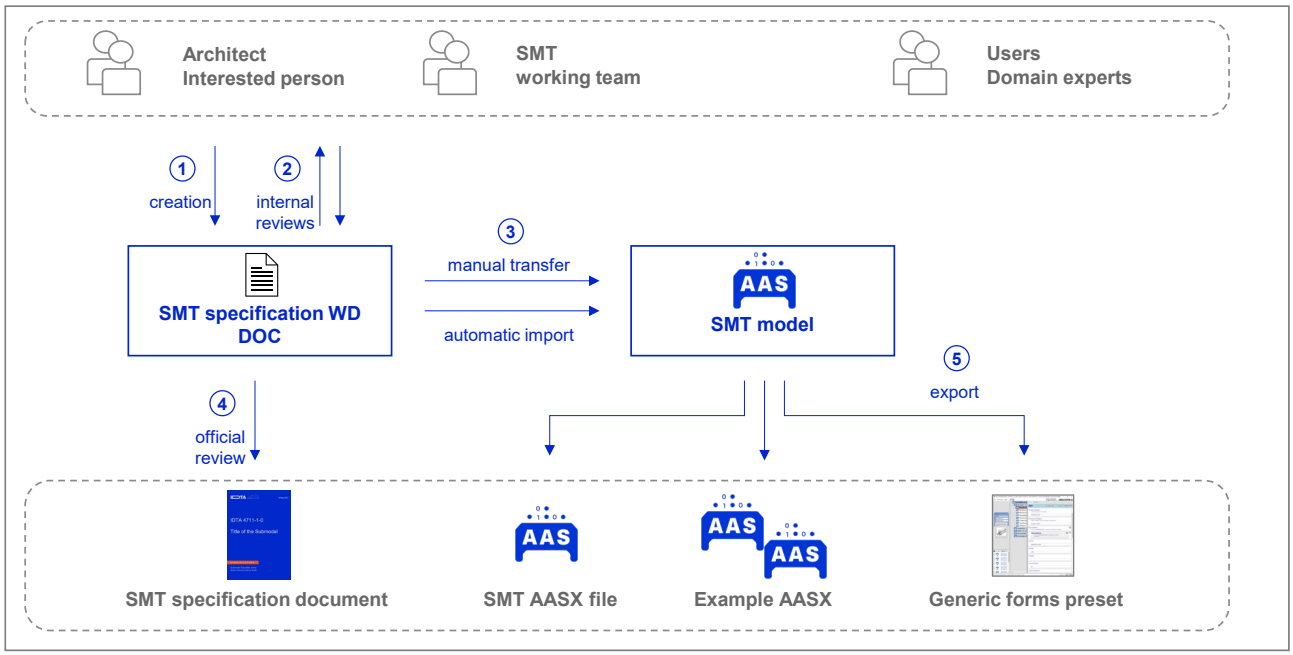

Figure 4 – Document driven workflow

The specification document is in the center of activities. The following working steps can be distinguished:

(1) Some member of the Submodel working team, e.g. the architect or the interested person [R1], create an initial version of the working draft (WD) of SMT specification document. This is a word process document (DOC), following the template of the IDTA.

The template is filled out with all structural relevant text sections as described in 2.2.

The requisites of the process "(5) Designing a Submodel b)" [R1] are considered.

(2) Frequent reviews within the working team are executed, using just the actual status of the working draft.

If more extensive reviews of the working draft are required, e.g. together with another working group, then a line numbered PDF is recommended and commenting via the IDTA comments template [R2] is recommended.

(3) If the discussions within the working team come to its conclusions and the "Designing a Submodel" phase [R1] is concluded, the SMT model, e.g. as AASX file, shall be created.

This can be done either by manually editing the SMT model, e.g. using the AASX Package Explorer [R3].

Or an (semi-) automatic import can be facilitated, e.g. by AASX Package Explorer, menu option "File / Import / Import Submodel from table".

(4) For approaching the official review, "(6) Review" phase [R1] is initiated, by preparing the final version of the working draft (WD) and handing over to architect and IDTA office.

These persons will follow on with the process, which will finally lead to a publication via IDTA GitHub and IDTA homepage.

(5) Using the SMT model, the final SMT AASX file can be defined. Examples can be generated using the action "Submodel / Turn to kind instance" [R3]. If adequate, a Generic forms preset can be done by defining the Qualifiers as in 9.3 and exporting via "File / Export / Export Submodel as options for generic forms" [R3]. This preset might be used to easily generate more AASX file examples. All these files shall be handed over to the architect and the IDTA office as well.

3.5 Model based workflow

The model-based workflow is suitable for complex subject matters and working teams looking deeply into particular aspects and capabilities of the resulting AASX model. Figure 5 demonstrates the workflow.

Note: The use of this workflow is deprecated. It was recommended standard for a period of time but is now replaced by the AsciiDoc workflows (see 3.7, 3.8).

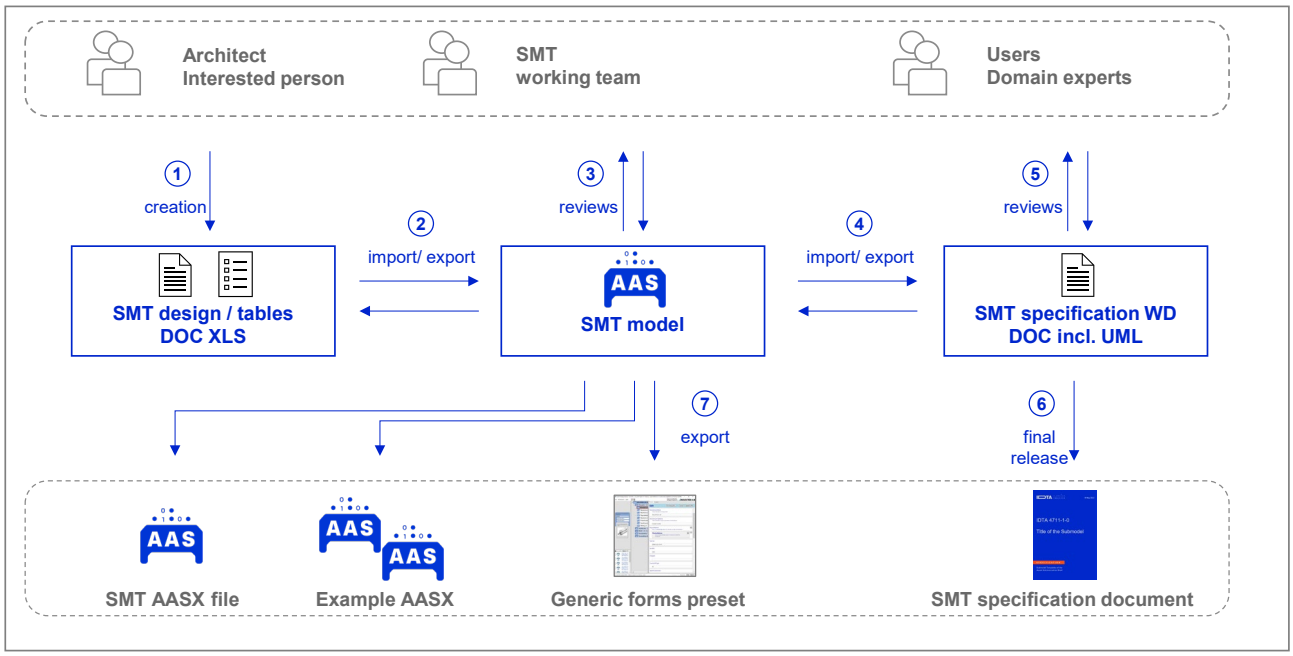

Figure 5 – Model based workflow

The SMT model is at the center of activities. The following working steps can be distinguished:

(1) Some members of the Submodel working team, e.g. the architect or the interested person [R1], create an initial design approach and partitioning into multiple tables of SubmodelElements.

It is also recommended to formulate the scope of the Submodel, e.g. during the kickoff the work team.

Table formats can be DOC or XLS[1], multiple tables can be parsed in a single document. These tables can also be easily reviewed by an external person, as well.

(2) Early in time, these tables are imported into the SMT model. The AASX Package Explorer [R3] provides such functions under "File / Import / Import from table", see [R5].

[1] Particularily, the Microsoft .docx and .xlsx formats can be used directly.

By the same mechanism, tables can be exported back via "File / Export / Export to tables" to form a SMT design roundtrip.

(3) Early reviews with the group can be executed directly by working on the SMT model.

The SMT model can be easily turned into example AASX by using the action "Submodel / Turn to kind instance" [R3] to test-drive the filling in with exemplary data.

Or, by defining Qualifiers as in 9.3 and using "File / Export / Export Submodel as options for generic forms" [R3], a Generic form preset can be generated, which could be handed to multiple people to test-drive the SMT.

Or, by using "File / Export / Export Submodel as snippet for PredefinedConcepts" [R3], source code for predefined concepts could be generated to be used for programmatic test exports of exemplary Submodel contents.

(4) Using the SMT model as a turntable, imports/ exports can also be done with respect to the working draft (WD) of SMT specification document.

Also, the scope and further definitions might be integrated already into the document.

The required tables can be directly exported [R3].

Additionally, UML can be generated by "File / Export / Export Submodel as UML" [R3] (see section 5).

(5) Using the SMT specification working draft (WD), together with the SMT model, reviews with the SMT working team, users and domain experts can be facilitated.

If more extensive reviews of the working draft are required, e.g. together with another working group, then a line numbered PDF is recommended and commenting via the IDTA comments template [R2] is recommended.

(6) For approaching the official review, "(6) Review" phase [R1] is initiated, by preparing the final version of the working draft (WD) and handing over to architect and IDTA office.

Architect and IDTA office will follow on with the process, which will finally lead to a publication via IDTA GitHub and IDTA homepage.

(7) Using the SMT model, the final SMT AASX file can be defined.

At least one Example AAS shall be generated, e.g. by using the action "Submodel / Turn to kind instance" [R3].

If adequate, a Generic forms preset can be realized by defining the Qualifiers as documented in section 8 and exported (see section 6.3). This preset might be used in turn to easily generate more Example AASX files.

All generated files shall be handed over to the architect and the IDTA office, as well.

3.6 Semantic driven workflow

The semantic based workflow is especially suitable for complex subject matters with no existing semantic definitions available for the subject under consideration. In the case of the semantic definition of a Submodel this also includes structural information. The workflow can also be applied in cases there is already a semantic definition available for the Submodel but in a different machine-readable format. Figure 6 demonstrates the workflow.

Note: This workflow can be combined with the AsciiDoc workflows (see 3.7, 3.8).

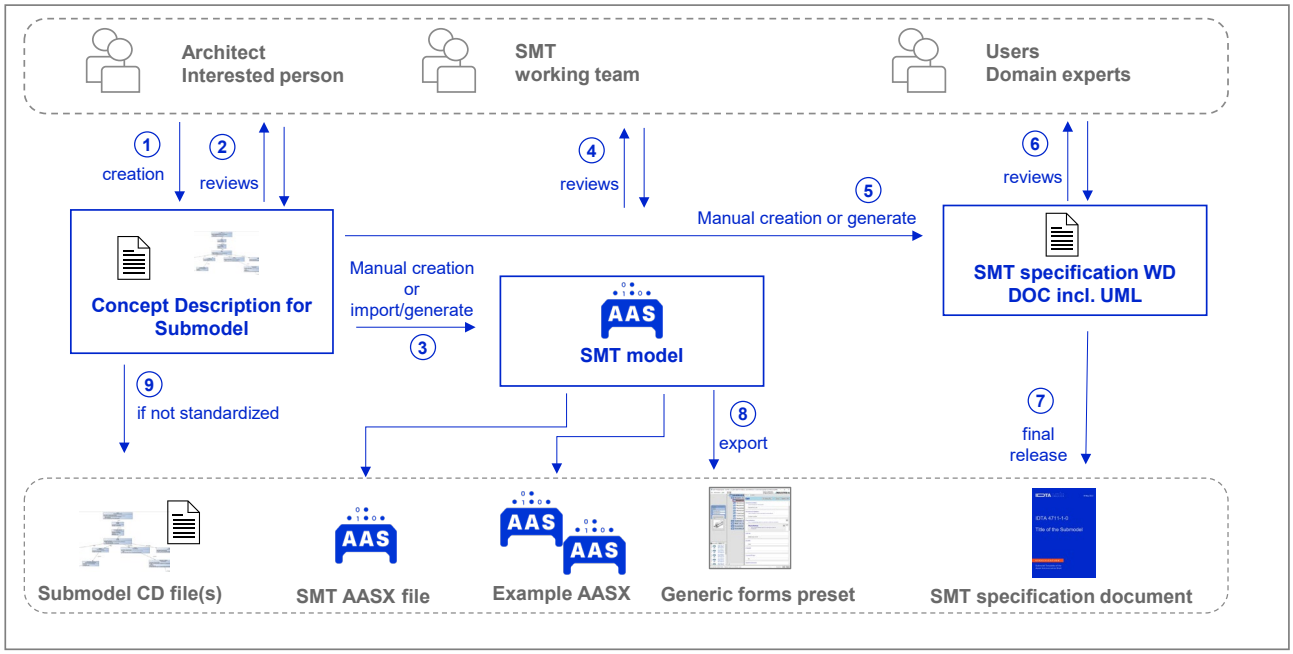

Figure 6 – Semantic driven workflow

The semantic definition of the Submodel itself is in the center of activities. A semantic definition of a Submodel is typically a model itself, in short such a model is sometimes called a semantic model. The following working steps can be distinguished:

(1) There are two ways to execute step (1):

(a) There is already an **existing open semantic definition** available for the Submodel under consideration in some standardized way, for example an application class in a dictionary like ECLASS or IEC CDD, or an existing W3C ontology or an existing semantic model of CATENA-X etc. In this context a semantic definition can only be directly used if this semantic definition has a globally unique identifier. Otherwise, if for example an IEC or ISO standard or an OPC UA companion specification exists but not with unique identifiers for the model then Step (b) needs to be followed.

(b) **No suitable open semantic definition is available** for the Submodel under consideration. In this case some member of the Submodel working team, e.g. the architect or the interested person [R1], creates an initial semantic definition or model in an appropriate format and with appropriate globally unique concept identifiers (semanticIds), e.g. SAMM [R9] or ECLASS fast track[2]. In a later stage external standardization of this Submodel template specific semantic definition in standards development organizations (SDOs) like ECLASS, IEC etc. can and should be planned.

[2] see https://www.eclass.eu/fileadmin/downloads/application-documents/ECLASS_terms-of-use_4-2_en.pdf

Note: This approach also might start with Excel or UML diagrams or any other supporting material before doing the first semantic definition of the Submodel, similar to what is described in chapter 3.5.

(2) Frequent reviews are executed with the domain experts and stakeholders.

(3) As soon as the semantic definition has a mature state, an AASX file is created. In the basic approach this is done manually. In an advanced approach a corresponding importer or generator is available that creates the AASX. Example: for ECLASS there is an importer available in the AASX Package Explorer that creates a corresponding Submodel. Semi-automatic approaches might also be supported.

(4) Internal reviews ensure that the created AASX file is correct.

(5) Then a corresponding textual specification of the Submodel template needs to be created. Again, this is either done manually, or this is generated (semi-)automatically.

(6) Again, internal reviews ensure that the created SMT Specification WD document meets the requirements.

(7) After the SMT Specification WD document is available, an official review can be started.

(8) If the SMT Specification can be released after the findings of the review are incorporated, the release is prepared containing the identified deliveries as specified (see section 3.2).

(9) If no existing open semantic definition for the Submodel under consideration is available (see step (1) b) then also the artifacts used to describe the semantic model in a machine-readable way – if available - shall be added to the release for future maintenance and reuse.

3.7 Workflow with GitHub based AsciiDoc working draft document

This workflow is an update of the model-based workflow (see section 3.5) using AsciiDoc technology for documentation purposes. Figure 7 demonstrates the workflow. Similar extensions to the document driven or semantic driven workflow can be made. They are not documented separately.

It is recognized that "AsciiDoc is a [suitable] plain text markup language for writing technical content" [R10]. With the use of AsciiDoc combined with GitHub functionality, an open, decentralized and traceable maintenance of the SMT specification becomes possible.

Note 1: This workflow is recommended, when multiple editors will edit the SMT specification on a decentralized basis and all concerned persons have backgrounds on source code management systems.

Note 2: This document gives a brief overview on this workflow and makes relevant specifications. To gain full overview, further material, such as demonstration or video training, is recommended.

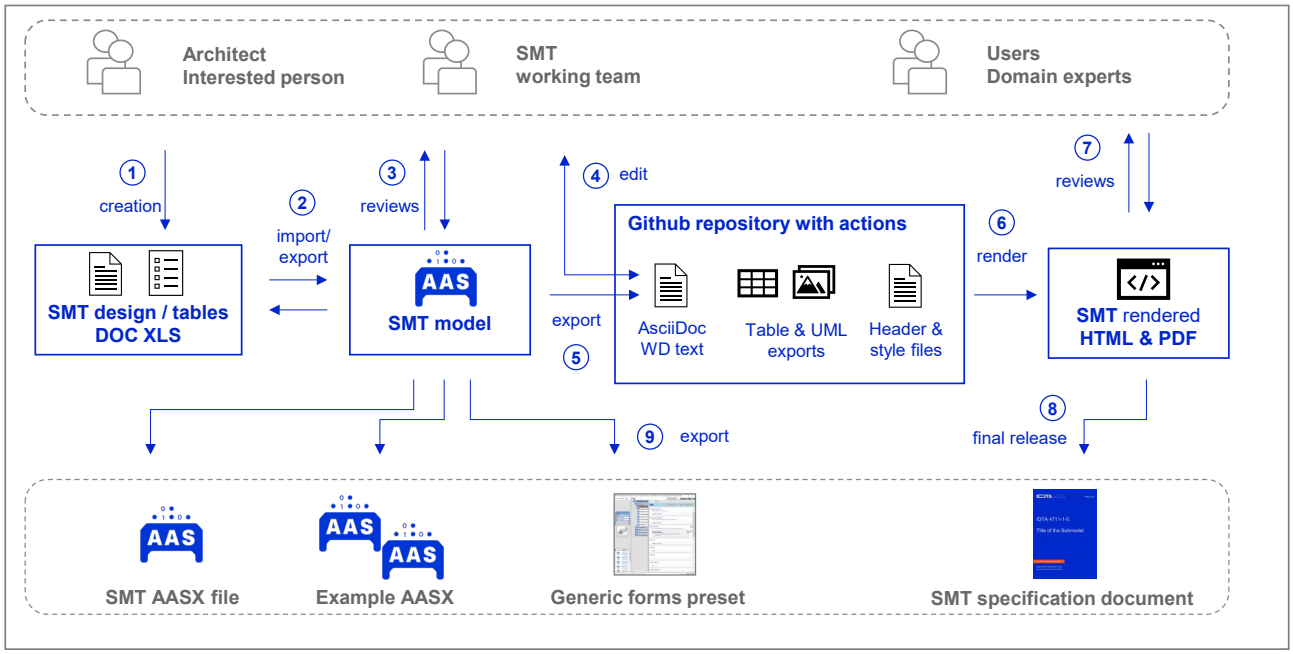

Figure 7 – Workflow with GitHub based AsciiDoc working draft document

The SMT model is at the center of activities. For the editing of the SMT specification WD, a GitHub repository with dedicated actions is used by all members of the SMT working team. This is already the case for the IDTA GitHub repository for Submodel templates [R11].

The workflow can start with some optional working steps:

(1) Some initial knowledge and modelling can be gathered and established by e.g. the architect or the interested person [R1] to create a start-up situation (optional).

(2) Well-formatted tables in DOC or XLS can be parsed into the SMT model (optional). The AASX Package Explorer [R3] provides such functions under "File / Import / Import from table", see [R5].

After, the work of the SMT working team is centered around the SMT model and the working draft in AsciiDoc:

(3) Early reviews on the SMT structure and definitions of SubmodelElements and ConceptDescriptions are performed directly on the SMT model as part of the SMT working team meetings (see section 3.5).

(4) Distributed editing of the SMT specification WD is done by accessing the individual related files on the GitHub repository. These files comprise:

- One or multiple AsciiDoc formatted WD text files. These contain e.g. the individual text bodies, chapters, paragraphs or hand-crafted tables in AsciiDoc markup language. By defining includes, the specification can be broken into parts, e.g. maintained by individual editors.
- Tables and UML diagrams. AsciiDoc has an extensive table syntax. There is a variety of free and/ or online tools to migrate from office formats to AsciiDoc or edit complex tables online. Many AsciiDoc tools also allow to include PlantUML files [R7] natively, so that UML diagrams can be specified and maintained as pure text files.
- Header files and style files. In order to bring individual parts together, use extensions, declare global options and more, header files can be used. For SMT specifications, suitable header files and style files shall be used [R13].

(5) Using the SMT model as a turntable, a large fraction of the SMT specification can be exported directly to AsciiDoc files via the AASX Package Explorer [R3].

The required tables can be exported by "File / Export / Export to tables" [R3].

Additionally, UML can be generated to PlantUML by "File / Export / Export Submodel as UML" [R3] (see section 5).

(6) Using pre-defined GitHub actions, the AsciiDoc files can be rendered to HTML and PDF files.

The following steps are the same as in the model-based workflow (see section 3.5).

(7) Using the SMT specification HTML and PDF renderings, together with the SMT model, reviews with the SMT working team, users and domain experts can be facilitated.

If more extensive reviews of the working draft are required, e.g. together with another working group, then a line numbered PDF is recommended and commenting via the IDTA comments template [R2] is recommended.

(8) For approaching the official review, "(6) Review" phase [R1] is initiated, by preparing the final version of the working draft (WD) and handing over to architect and IDTA office.

Architect and IDTA office will follow on with the process, which will finally lead to a publication via IDTA GitHub and IDTA homepage.

(9) Using the SMT model, the final SMT AASX file can be defined.

At least one Example AAS shall be generated, e.g. by using the action "Submodel / Turn to kind instance" [R3].

If adequate, a Generic forms preset can be realized by defining the Qualifiers as documented in section 8 and exported (see Clause 6.3). This preset might be used in turn to easily generate more Example AASX files.

All generated files shall be handed over to the architect and the IDTA office, as well.

3.8 Model based workflow with single source AsciiDoc working draft document

This workflow emphasizes the SMT model as a single source of truth for the SMT specification. Figure 8 demonstrates the workflow. Some advantages of the GitHub based AsciiDoc editing (see section 3.7) are sacrificed in order to keep all elements of the specification process together and to allow an automatic referencing/ including of all exporting/ generation steps. This leads to a short round-trip time for rendering consistent SMT previews and easy maintainability after publication.

Note 1: This workflow is recommended, when efficient and future oriented maintenance of the SMT specification is applicable.

Note 2: This document gives a brief overview of this workflow and makes relevant specifications. To gain a full overview, further material, such as demonstration or video training, is available by IDTA.

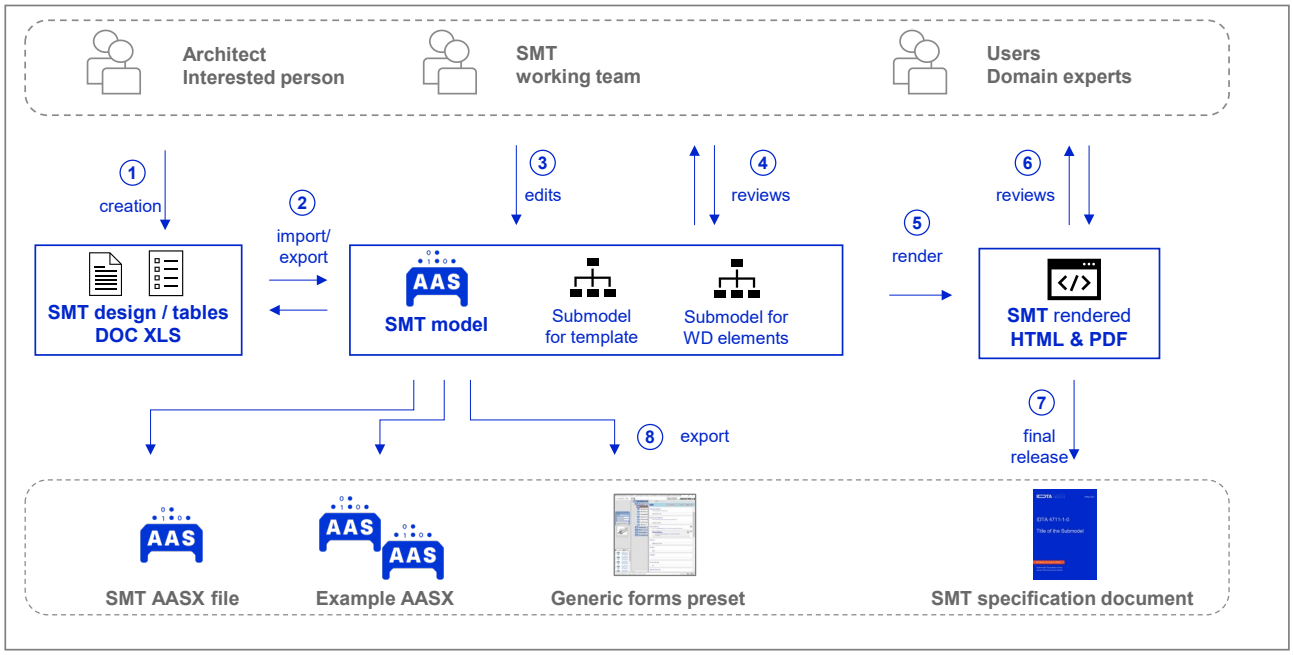

Figure 8 – Workflow with single source AsciiDoc working draft document

The SMT model is at the center of activities. It contains two Submodels: one Submodel stands for the Submodel template and Submodel template elements. One Submodel stands for the human readable parts of the SMT specification, including also structure, images, figures, header and style files.

The workflow can start with some optional working steps (same as described in section 3.5 and section 3.7):

(1) Some initial knowledge and modelling can be gathered and established by e.g. the architect or the interested person [R1] to create a start-up situation (optional).

(2) Well-formatted tables in DOC or XLS can be parsed into the SMT model (optional). The AASX Package Explorer [R3] provides such functions under "File / Import / Import from table", see [R5].

After, all subsequent editing/ update steps occur only in the SMT model:

(3) By using the AASX Package Explorer [R3] or similar tooling, the architect and SMT working team members edit/ update/ review the SMT model. When using AASX Package Explorer [R3], this step is monolithic and no concurrent use of the associated AASX file is possible.

The editing of the human-readable part is also facilitated by the AASX Package Explorer. By using specific semanticIds and Extensions (see section 9.6), the following features are accomplished:

- **Blobs for markup of chapters and sections** of the SMT specification document. By using dedicated semanticIds for different levels of headings, suitable markup is automatically generated (indexing possible). The hierarchy level and idShorts of the Blob elements can be used to structure the document visually in the AASX model.
- **Blobs for markup of single or multiple text blocks** in AsciiDoc. Via the multi-line editing option of the AASX Package Explorer, large chunks of AsciiDoc markup can be edited in an efficient way.
- The architect/ SMT working team may decide to use a **fine-grained structure** (Blobs for each individual heading and text block) or **coarse-grained structure** (large fractions of the AsciiDoc markup in one single Blob element).
- **Blobs to include further content**, which will be provided as single files in the AsciiDoc generation process. This might include **images/ figures, style files, header files** and more. Via Extensions it can be controlled, if 'include' markup is generated to automatically include these files or not. AASX Package Explorer includes a "blob assistance" to read images and convert to BASE64 encoded text, making it unnecessary to always define supplemental files.

- **File elements** and supplemental files to provide further content, e.g. **large images/ figures or markup files**. These content files can be automatically included, as well. AASX Package Explorer allows editing text files directly within the package.
- **References to automatically generate the table format** for Submodels and SubmodelElements (see section 7). By creating a ReferenceElement to a structure of SubmodelElements and assigning an appropriate semanticId, large fractions of the table content of the SMT specification can be generated automatically.
- **References to automatically generate UML** (see 5.4). By creating a ReferenceElement to a structure of SubmodelElements and assigning an appropriate semanticId, UML code in PlantUML [R7] can be automatically generated. Via Extensions, the style and level of depth for UML can be controlled and classes could be suppressed, in order to always present the intended content to the reader.

The linear sequence of the above elements in the Submodel for the SMT specification specifies also the sequence of generation/ including the generated markup.

(4) Early reviews on the SMT structure and definitions of SubmodelElements and ConceptDescriptions of the template and of the human-readable parts are performed directly on the SMT model as part of the SMT working team meetings (see section 3.5).

(5) AASX Package Explorer [R3] is able to bundle all required files for the rendering of AsciiDoc HTML and PDF together. By calling "File / Export / Export / Export Submodel as AsciiDoc SMT spec ..", a temporary working directory will be created, all required files (markup, tables, UML, images, headers and styles, ..) will be generated/ copied into this working directory and the files will be zipped in order to provide one coherent snapshot, which can be provided easily to the users (architect, SMT working team, domain exports, publication).

Optionally, the application is able to execute a configurable script allowing to run docker containers/ actions, which render the exported files to HTML/ PDF. This allows producing and viewing HTML/ PDF contents of the actual SMT specification very quickly.

The following steps are the same as in the other model-based workflow(s) (see section 3.5 and 3.7).

(6) Using the SMT specification HTML and PDF renderings, together with the SMT model, reviews with the SMT working team, users and domain experts can be facilitated.

(7) For approaching the official review, "(6) Review" phase [R1] is initiated, by preparing the final version of the working draft (WD) and handing over to architect and IDTA office.

Architect and IDTA office will follow on with the process, which will finally lead to a publication via IDTA GitHub and IDTA homepage.

(8) Using the SMT model, the final SMT AASX file can be defined.

At least one example AAS shall be generated, e.g. by using the action "Submodel / Turn to kind instance" [R3].

If adequate, a Generic forms preset can be realized by defining the Qualifiers as documented in section 8 and exported (see section 6.3). This preset might be used in turn to easily generate more example AASX files.

All generated files shall be handed over to the architect and the IDTA office, as well.

4 SMT dropins

4.1 Definition

A SMT dropin is a set of SubmodelElements and associated ConceptDescriptions, which can be used in multiple SMT specifications. The intended purpose is re-use.

A SMT dropin is not a Submodel template on its own, i.e. there is no Submodel guided by a SMT dropin.

A SMT dropin is a logical model (compare SMT model, see Clause 3.2). A SMT dropin is always part of a Submodel, the reason being that the same technology and APIs as for SMT in general can be used.

Note: idShort names might be changed as well as shown in the example in Figure 9.

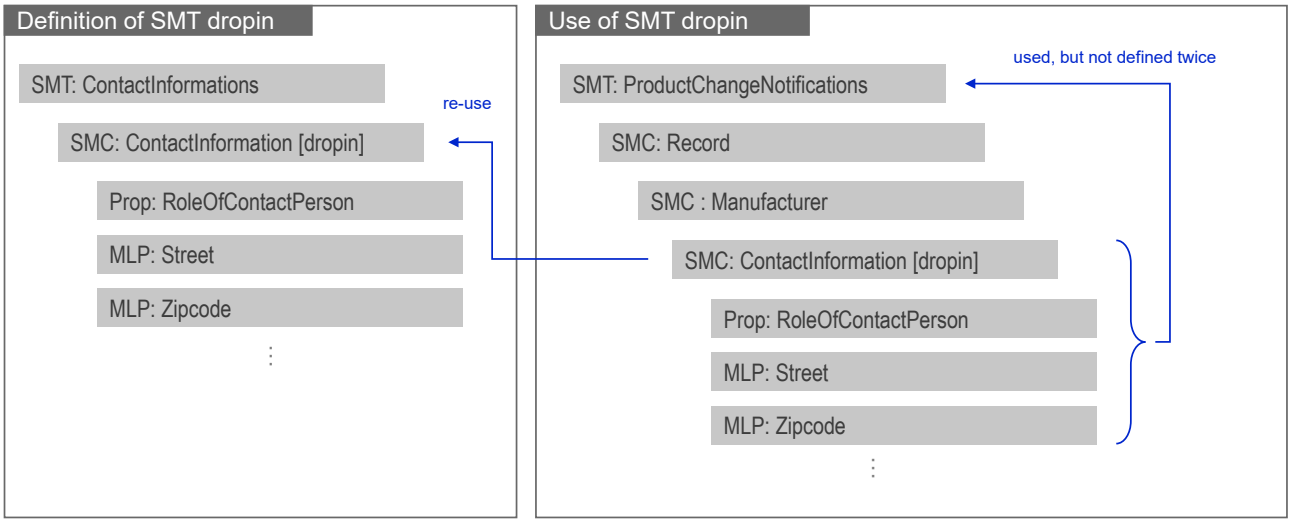

Figure 9 – General approach for definition and use of SMT dropins

4.2 Process

For defining SMT dropins, the same process is used as for a Submodel template. This document specifies the definition and application of SMT dropins, as well.

4.3 Organization of model elements

Definition of SMT dropins might occur in different organization schemes (depending on the implementation by e.g. IDTA or other consortia dedicated to a specific domain).

(1) For a limited amount of time, a SMT dropin might be defined in the context of a regular Submodel template, until it is decided to be organized in a dedicated way (see Figure 10 (a)). This has the additional benefit of an owner/ caretake to be already assigned.

(2) For executing the process, the SubmodelElements and ConceptDescriptions of a SMT dropin might be organized in a dedicated SMT model (see Figure 10 (b)). This makes handling the model elements simple and effective during the process execution. This may start after discovery of potential re-use of contents of a Submodel template. This is recommended practice for the SMT process.

(3) For long-term maintenance, the SubmodelElements and ConceptDescriptions of a SMT dropin might be transferred to a repository and the SubmodelElements of the SMT dropin might be organized by some Submodel template(s) within the repository (see Figure 10 (c)).

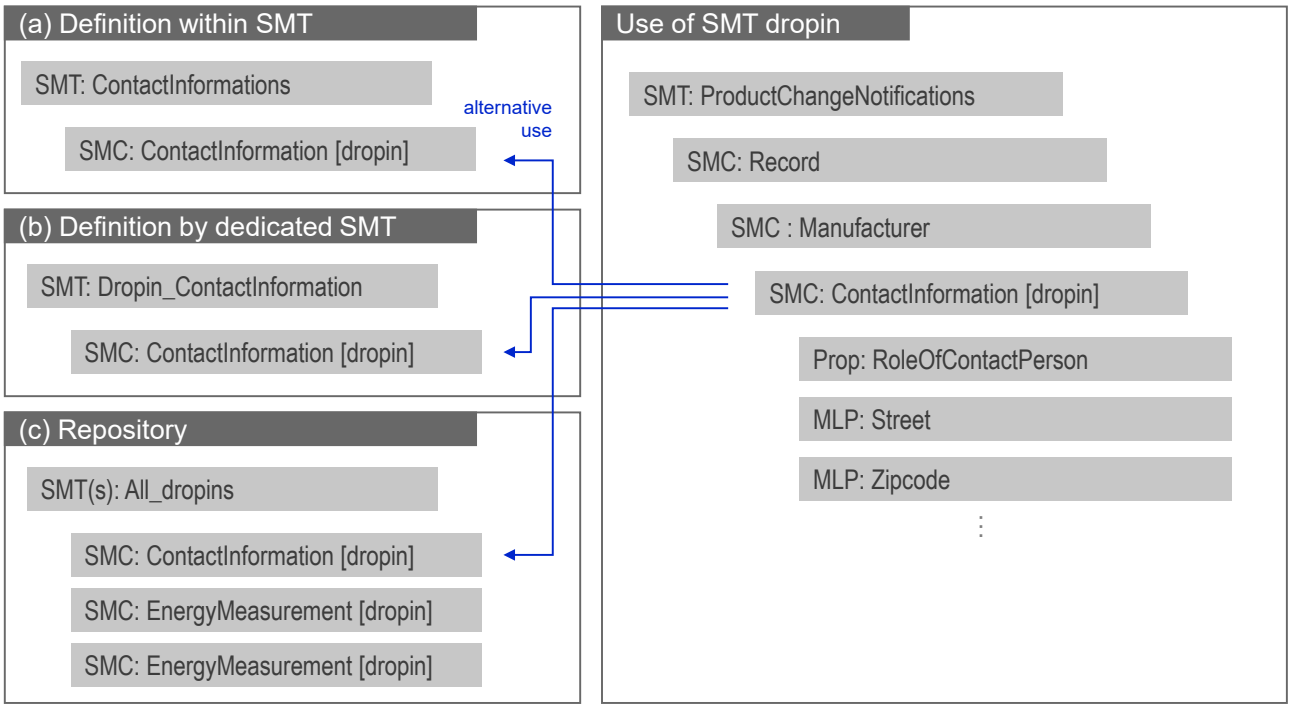

Figure 10 – Alternatives for organizing SMT dropins

4.4 Handling of semanticIds

The definition and the use of a SMT dropin makes use of the same semanticId. In order to formally distinguish between these two cases, adding of the following supplementalSemanticIds is required (see Table 4, Figure 11, Figure 12):

Table 4 – SupplementalSemanticIds for SMT dropins

SupplementalSemanticId Description	Examples
https://admin-shell.io/smt-dropin/**smt-dropin-definition**/1/0 **Specifies** the associated semanticId of this SubmodelElement to be the root of the definition of an SMT dropin. An AAS user application might use this information to lookup a list of possible SMT dropins.	n/a
https://admin-shell.io/smt-dropin/**smt-dropin-use**/1/0 **Marks the use** of a SMT dropin and identifies the root of theSubmodelElements of the SMT dropin. The human reader or the machine is informed, that there is a SMT dropin and that the definition of this and the dependent SubmodelElements is not required to be in the same SMT specification.	n/a

Note: If the model elements of a SMT dropin are organized by a dedicated Submodel, this Submodel might have a dedicated semanticId. However, this semanticId is independent from the semanticId of the root SubmodelElement of the SMT dropin.

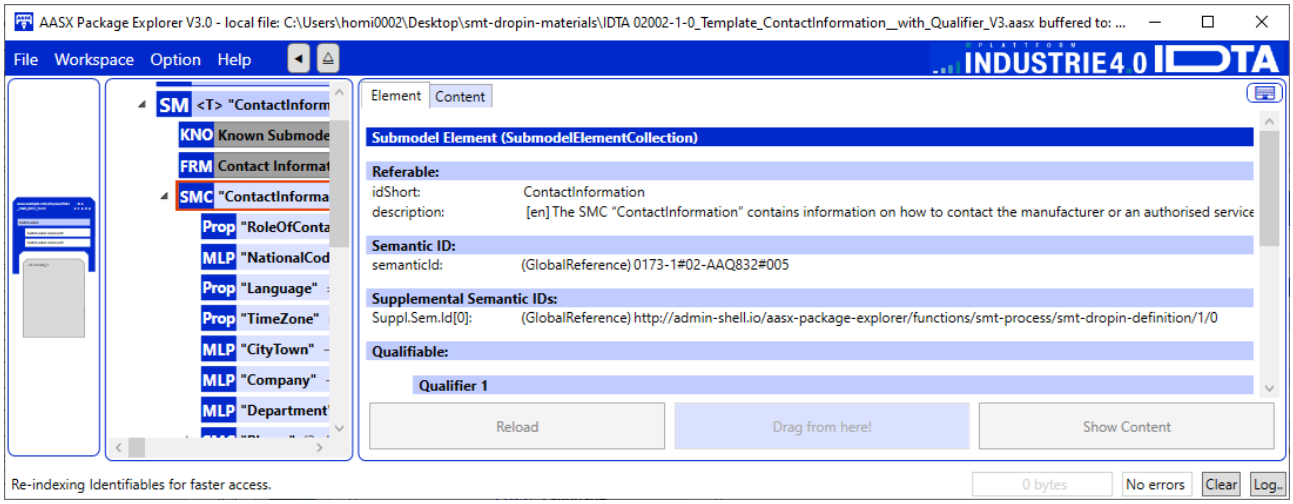

Figure 11 – Example for SMT dropin definition

4.5 Publication of SMT dropins

For the publication of SMT dropins, a dedicated folder is foreseen in the IDTA GitHub repository for Submodel templates [R11]. For each SMT dropin, a sub-folder is foreseen with at least these artifacts (see 3.2, Table 5):

Table 5 – Artifacts for SMT dropins

Artifact	Description	Cardinality
SMT specification	Either a dedicated SMT specification or a SMT specification defining this SMT dropin (see Figure 10, (b) or (a)) shall be given. In case of SMT specification (b) defining this SMT, this would be a copy of the existing specification.	One
Pure SMT AASX file	Either a dedicated AASX file or an AASX file defining this SMT dropin shall be given.	One
Qualified SMT AASX file	Either a dedicated AASX file or an AASX file defining this SMT dropin shall be given.	ZeroToOne

4.6 Indication of usage of SMT dropins

A SMT specification, which takes advantage of a SMT dropin, shall indicate this usage to the reader of the specification.

(1) In the SMT model, the **supplementalSemanticId** shall be set (see section 4.4) for the root SubmodelElement of the SMT dropin (see Figure 12).

(2) When creating or generating **tables** (see section 7), a **note** shall be added to the root SubmodelElement of the SMT dropin (see Figure 13).

(3) The use of the SMT dropin shall also be added to the section "**Relevant standards**".

(4) If technically possible, the **overview UML diagram** of the SMT specification should indicate the use and the source of the SMT dropin by means of a UML stereotype «smt-dropin-use» (see Figure 14).

Note to (2): This can be accomplished e.g. by setting the description of the SubmodelElement accordingly.

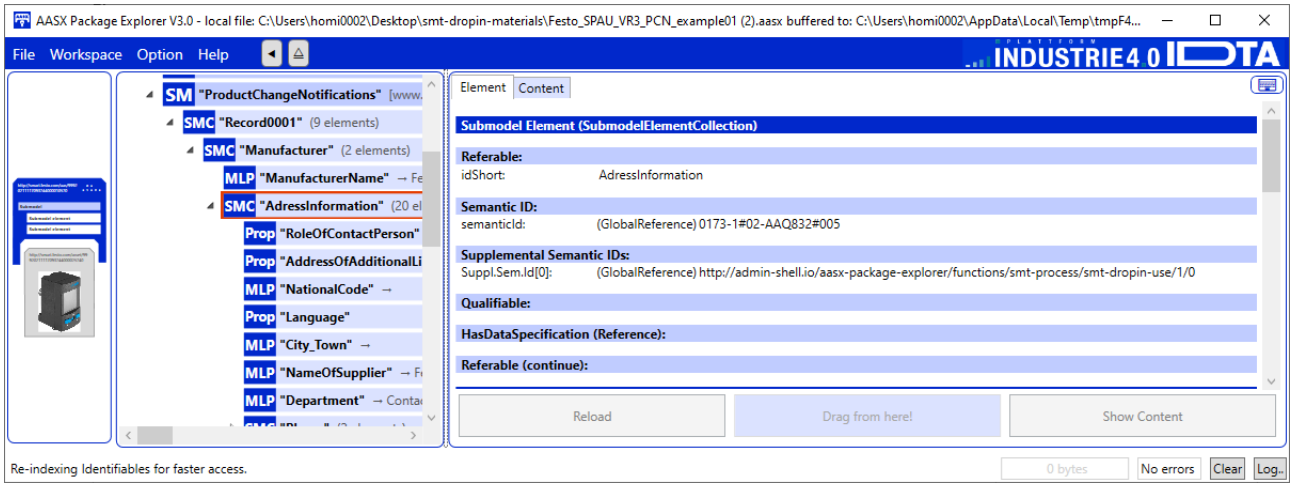

Figure 12 – Example for use of SMT dropin in SMT model

idShort	Description@en	example	
[MLP] ManufacturerName	[IRDI] 0173-1#02-AAO677#002 legally valid designation of the natural or judicial person which is directly responsible for the design, production, packaging and labeling of a product in respect to its being brought into circulation	[langString] Beispiel & Söhne@DE	1
[SMC] AdressInformation	[IRDI] 0173-1#02-AAQ832#005 Address information of a business partner Note: this set of information is a SMT dropin named "Contact Information", semanticId: 0173-1#02-AAQ832#005.	n/a	1

Figure 13 – Example for use of SMT dropin in SMT specification tables

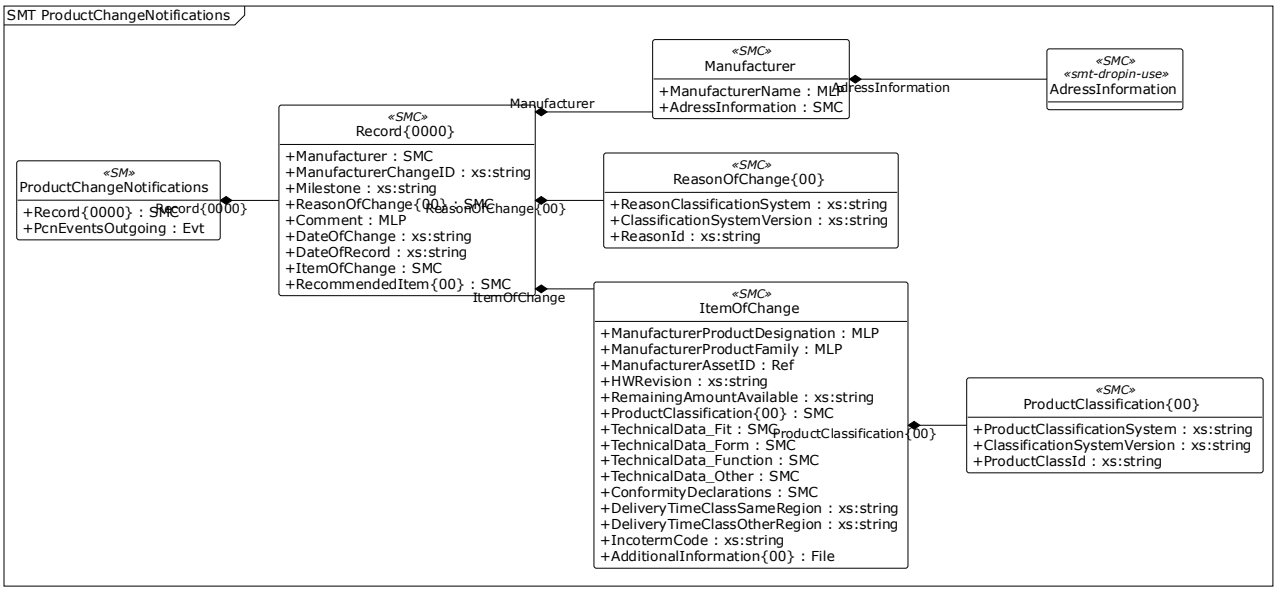

Figure 14 – Example for use of SMT dropin in UML by UML stereotype (recommended)

4.7 Degrees of flexibility when using a SMT dropin

Certain degrees of flexibility can be achieved by the Submodel template using a SMT dropin definition, if the use of a SMT dropin is indicated by the respective supplementalSemanticId for "use" (see 4.4). As an example, in Figure 13 and Figure 14, the root of the contact information was decided to be titled "AddressInformation". The following degrees of flexibility are (see Table 6):

Table 6 – Degrees of flexibility when using a SMT dropin

Flexibility	Description
Change of idShort	The idShort of the SubmodelElement using a SMT dropin might be changed, e.g. if the common wording of the usage domain motivates this. Recommendation is to take over the idShort from SMT dropin, if possible.
Change of description	The description of the SubmodelElement using a SMT dropin might be changed, if the idShort was changed, as well. Recommendation is to take over the description from SMT dropin, if possible.
Change of cardinality	The cardinality of the SubmodelElement using a SMT dropin (given by a Qualifier, see 9.2) might be changed.
Change of semanticId	The semanticId of the SubmodelElement using a SMT dropin identifies the SMT dropin and shall not be changed, unless a ConceptDescription in the AASX file or repository exists, which declares an "isCaseOf" with the semanticId of the respective SMT dropin. Note 1: Although tempting from semantic point of view, this solution is not recommended, as technical frameworks handling the AAS might not be aware if this "isCaseOf" relation. Note 2: In such cases, consider using the "original" semanticId and adding a supplementalSemanticId enriching the meaning or distinguishing between multiple uses in the collection of SubmodelElements. Note 3: If the semanticId of the SubmodelElement using a SMT dropin is changed and the supplementalSemanticId for "use" according 4.4 is given and no ConceptDescription with "isCaseOf" relation is properly given, then this is seen as modelling error. Note 4: The same rules hold for changing the "semanticIdListElement" within a SubmodelElementList.
Change of category	Since the category is deprecated, the category can be omitted or changed.
Change of display name	The display names may be changed; additional display names in other languages might be added.
Change of extensions	Existing extensions shall not be removed. However, it is allowed to add additional extensions.

For all other attributes, i.e. other template qualifiers and data specifications defined for the dropin as well as attributes like annotations for AnnotatedRelationshipElement or variables in operations, the value shall be identical to the one defined for the dropin. The same holds if some attributes have a fixed predefined value.

5 UML generation

5.1 General

Having an SMT model available, Unified Modeling Language (UML) can be generated easily via the AASX Package Explorer. This is achieved using "File / Export / Export Submodel as UML" [R3].

Precondition is, that cardinalities of the different SubmodelElement are designated via the Qualifier "Multiplicity", according to section 9.2. UML can be generated for Submodels of kind = Template or kind = Instance, even with (example) values attached.

5.2 UML design style

Unified Modeling Language (UML) is a highly specified, general-purpose, modeling language in the field of software engineering and is used for many purposes. One purpose is to illustrate the structure of AAS elements given by a SMT (SMT) specification. For this purpose, within such specification, the following provisions are set:

(1) The structure of AAS element shall be represented by a class diagram, which is titled according to the name of the SMT specification.
(2) AAS elements providing children (such as SubmodelElementCollection, Operation and more) shall be represented by UML class elements, with the AAS element type or its abbreviation according to [6] set as stereotype.
(3) Such children (of complex elements such as SubmodelElementCollection, Operation and more) shall be represented as attributes within the respective UML class.
(4) Such attributes shall be marked as public ('+') and shall feature the AAS element type or data type or its abbreviation according to [6].
(5) If the cardinality (multiplicity) of such an attribute is other than '[1]', one of the following cardinality notions shall be used: [0..1], [0..*], [1..*].
(6) If such attribute represents an AAS element providing children (such as SubmodelElementCollection, Operation and more), and this AAS element is represented by the respective class diagram as well, then an aggregation association (filled diamond arrow) shall be expressed, with the name of the AAS element given by the association and its cardinality at the 'part' end of the association.

5.3 UML via XMI export

Figure 15 shows the result of export UML to XMI 2.1 format via the AASX Package Explorer. The XMI does not contain the design of a diagram, so this must be done using an UML authoring tool, such as Enterprise Architect. An intended layout can be achieved.

Note 1:	As of Jan 2022, the directionality is not exported correctly. So, the direction of each association needs to be set to unspecific, in order to render aggregation associations in a correct way.
Note 2:	The UML authoring tool should be able to export vector graphics in order to allow good scaling for publication. In Enterprise Architect this is achieved by "Start / Preferences / General / Clipboard Format: Metafile" and "Publish / Save Image / Save to Clipboard" (V13).
Note 3:	The shown example required about 10 min design time.

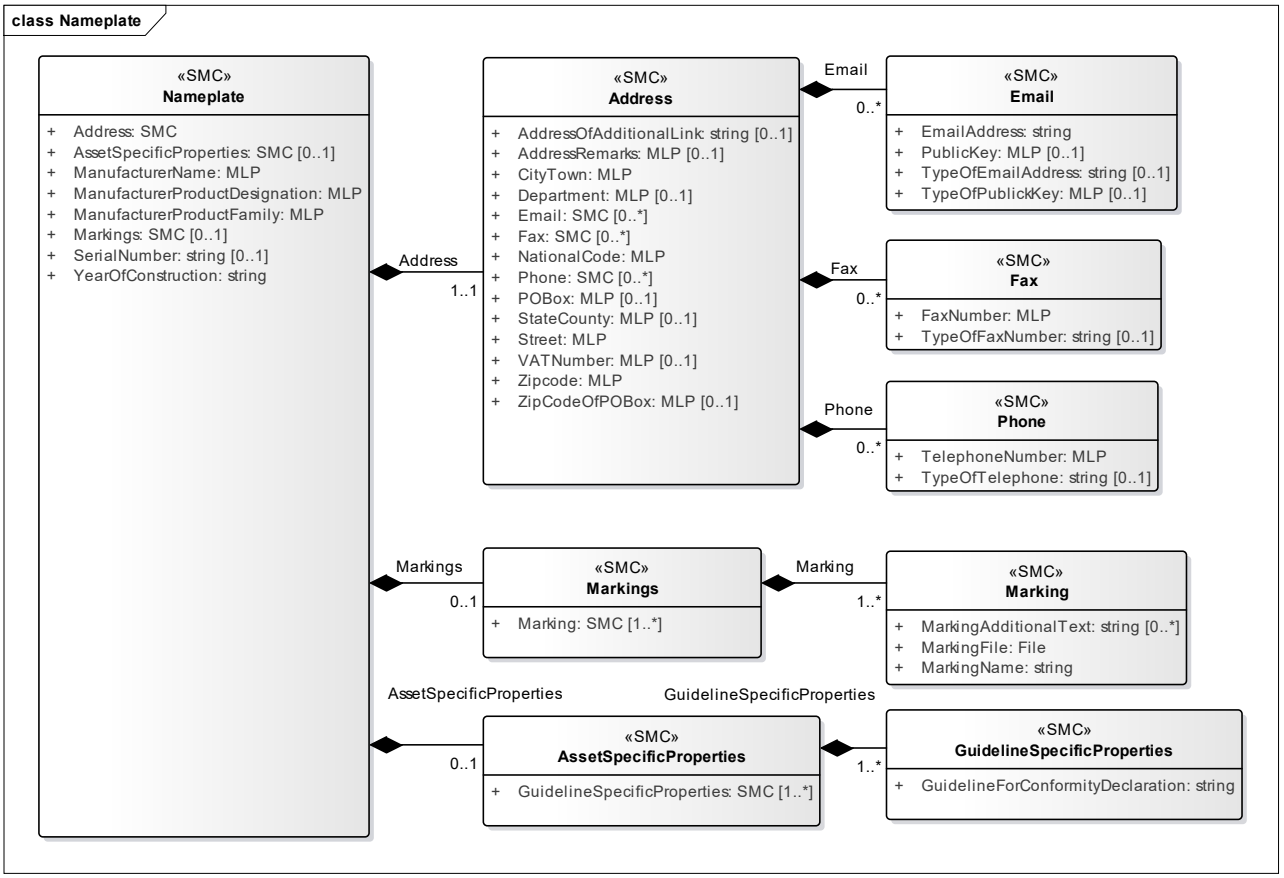

Figure 15 – Example UML generation by exporting XMI and manual layout in UML authoring tool

5.4 UML via PlantUML export

Figure 16 shows the result of exporting UML to PlantUML format via the AASX Package Explorer. The file contents can be copied to the clipboard, as well, and directly pasted to [R7].

Note 1: The generated UML can be slightly adjusted, but not manually laid out, by some options. In the example, the line "mainframe SMT Nameplate" introduced a frame around the diagram.
Note 2: The example file was saved as SVG, which allows vector scaling for publication.
Note 3: The shown example required about 15 sec design time.

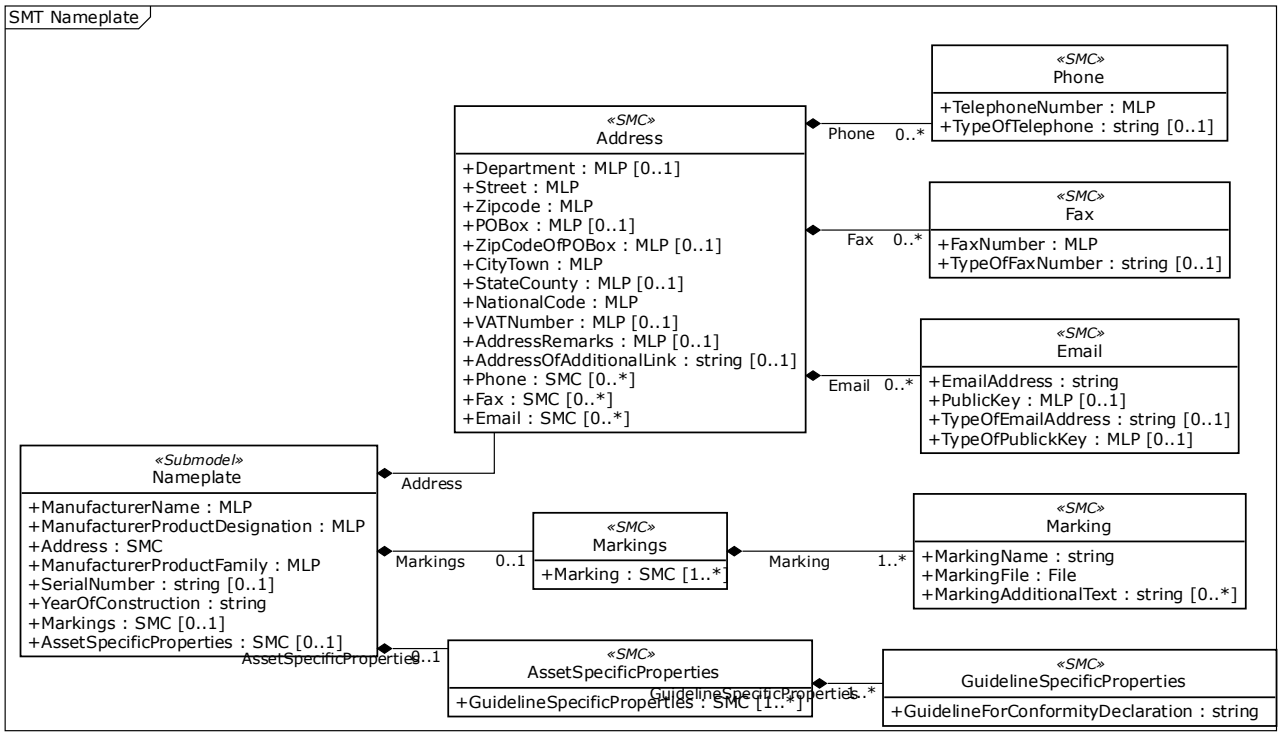

Figure 16 – Example UML generation by exporting to PlantUML and automatic layout

6 Generic forms preset

6.1 General

The AASX Package Explorer [R3] allows using the plugin AasxPluginGenericForms to assist users in creating Submodel instances based on SMTs (see Figure 17). The plugin displays a visual assistant to easily create a Submodel instance according to a SMT and fill out this instance with data. This allows also non-expert users to provide subject matter information to the AAS.

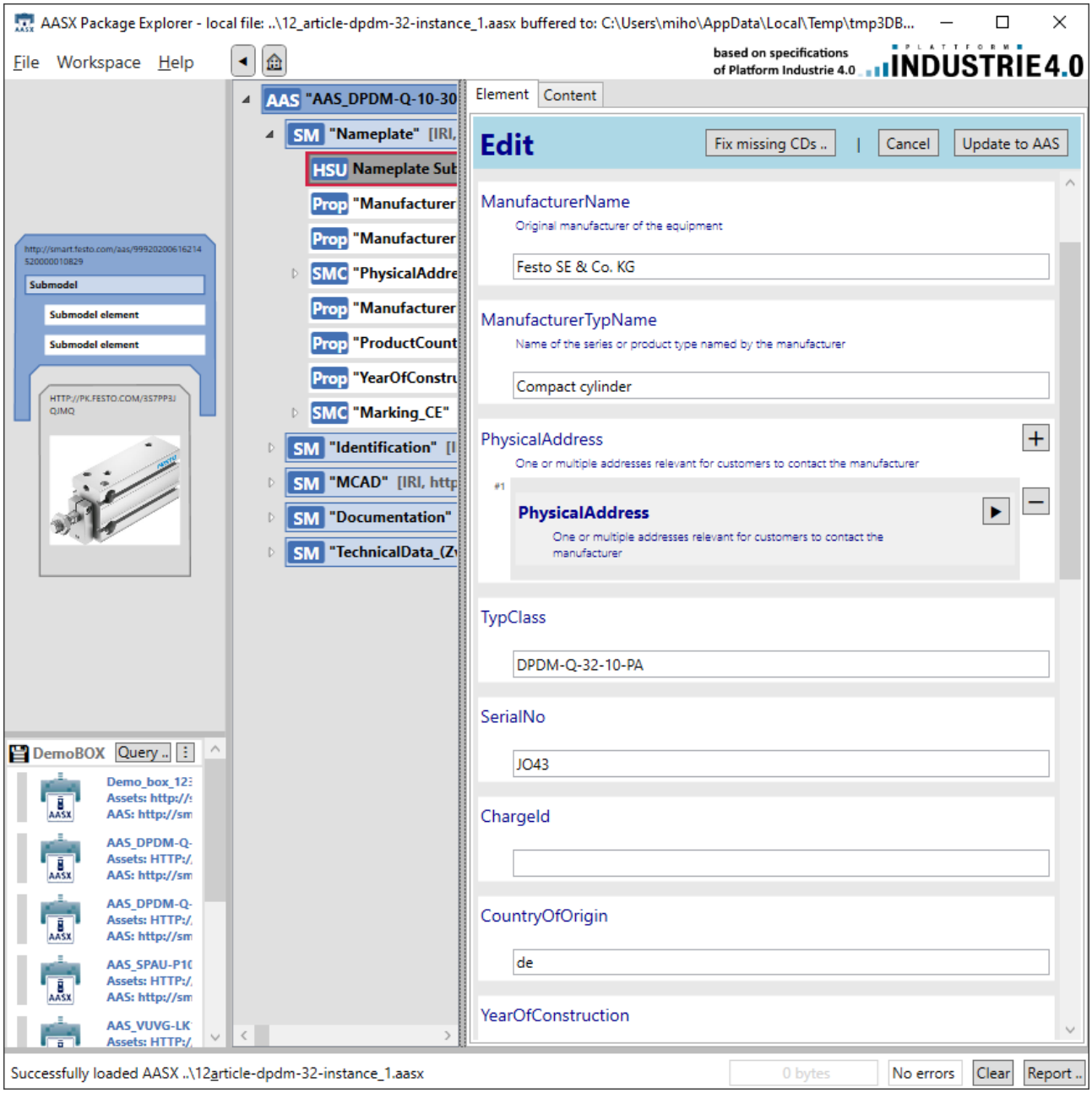

Figure 17 – AASX Package Explorer offering the easy filling out of Submodel "Nameplate"

6.2 Attribution of a SMT

The different SubmodelElements of a SMT need to be attributed by Qualifers in order to provide enough information for an adequate rendering of generic forms. These Qualifiers are described in section 9.3.

6.3 Exporting options file

The SMT might be exported via "File / Export / Export Submodel as options for generic forms" [R3]. This leads to the creation of a file "*.add-options.json", which can be used by other users to automatically generate and fill out Submodels.

6.4 Usage of generic forms in AAS user applications

In order to use a generic forms preset in AASX Package Explorer [R3], the user needs to locate the ".\plugins\AasxPluginGenericForms" folder and copy the respective "*.add-options.json" file into this folder (see Figure 18). After re-start of the application, using "Workspace / Plugins / New Submodel", a new Submodel can be created.

The architect and IDTA office will also check if the Submodel and its generic forms preset can be included into the standard deployment of the AASX Package Explorer.

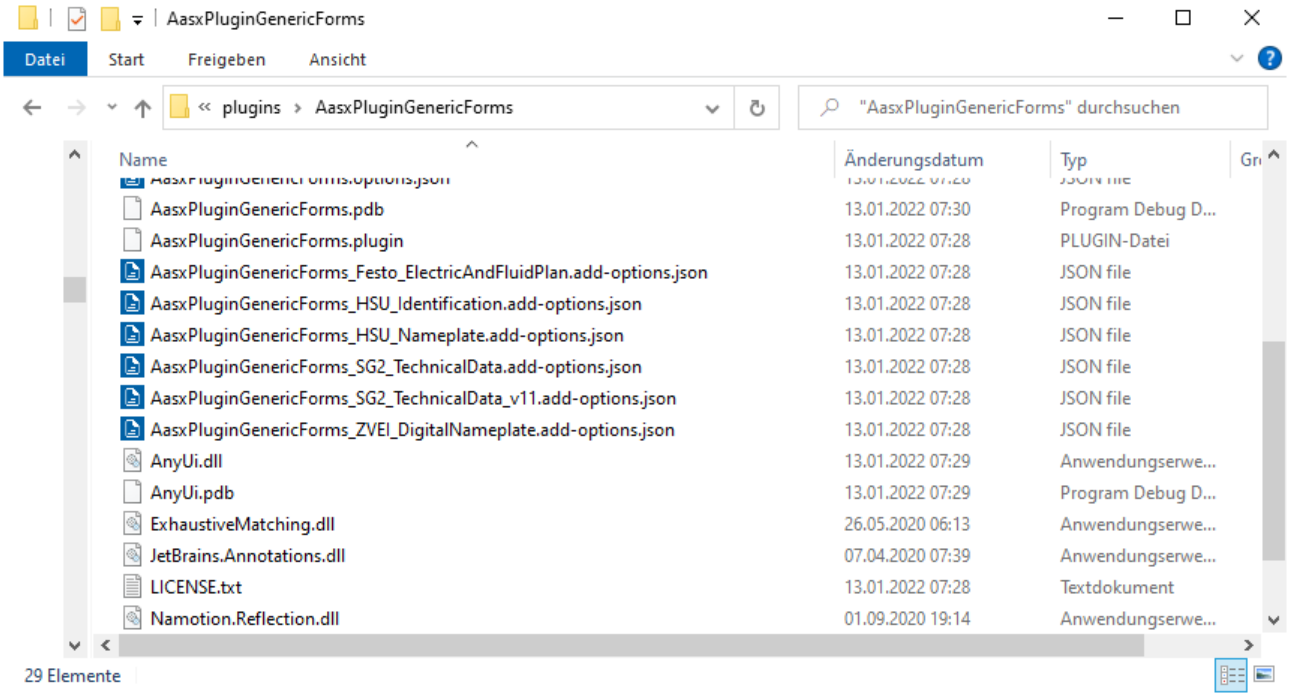

Figure 18 – Plugin folder for Generic forms

7 Table format for Submodels and SubmodelElements

7.1 General

The SMT document shall provide a set of tables for the Submodels and SubmodelElements, which are specified by the SMT. In the SMT, these AAS elements shall be of kind = Template. The tables are designed to provide an overview of these AAS elements, allowing all stakeholders (see section 1.3) to understand the design of the particular SMT.

The described table format is not self-explanatory. Each SMT document shall feature an appendix, explaining the table format (see Annex A). This appendix is provided by the template document of the IDTA.

7.2 Table heads

The SMT document shall feature a separate table for each AAS element, which is specified by the SMT and which hierarchically specifies child AAS elements (consequently named "AAS element with children"). Typically, but not exclusively, these are Submodels and SubmodelElementCollections (see section 1.6), but can be also Operations or Entities.

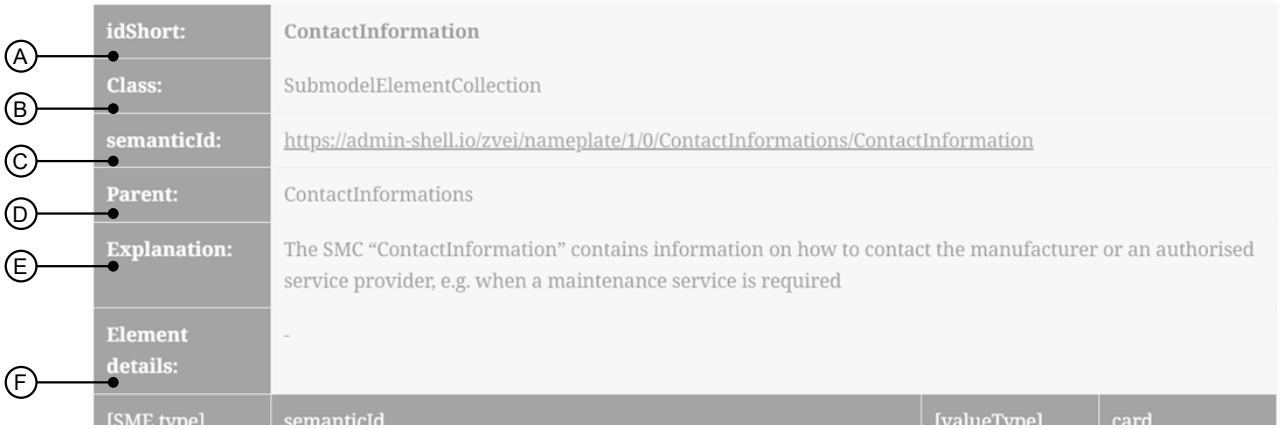

Figure 19 – Format of table heads

Figure 19 shows the format of the heading part of such table. For the different rows, the following provisions are given by Table 7:

Table 7 – Format of table heads

ID	Label	Provision
A	idShort:	This cell shall contain the idShort of the AAS element with children. Could be an idShort with enumeration template such as "__000__" or "__00__" (see section 9.4). If the AAS element is used multiple times as child of a parent AAS element, then a list of comma separated idShorts can be given. Must not contain further information, such as "Note: the above idShort shall always be as stated."
B	Class:	This cell shall contain the AAS element type of the given AAS element with children. Can be: Submodel, SubmodelElementCollection, Operation, Entity, AnnotatedRelationshipElement.
C	semanticId:	This cell shall contain the idShort of the AAS element with children.
D	Parent:	This cell shall contain the idShort of the parent of the AAS element with children. If multiple parents in the SMT use it, then a list of comma-separated idShorts can be given. Must not contain further information, such as notes or annotations.
E	Explanation	This cell shall contain an explanation, which should be identical to the English language of the description of the AAS element with children.
F	Element details	Attributes of the meta model element (e.g. SML, SMC), such as orderRelevant or typeValueListElement.

Note: It is strongly recommended to use exactly the rows and columns as described. The import tools of AASX Package Explorer rely on this.

7.3 Table row items

Each table for AAS elements with children features multiple row items, one per child of the AAS element. Such row item shall be exactly one table row in the word processor document. In order to accommodate sufficient information for the stakeholders, each cell of the row item might contain multiple information, delimited by line breaks.

Note: This row format simplifies the manual editing of tables, such as described in the document driven workflow (see section 3.4).

Note: It is strongly recommended to use exactly the rows and columns as described. The import tools of AASX Package Explorer rely on this.

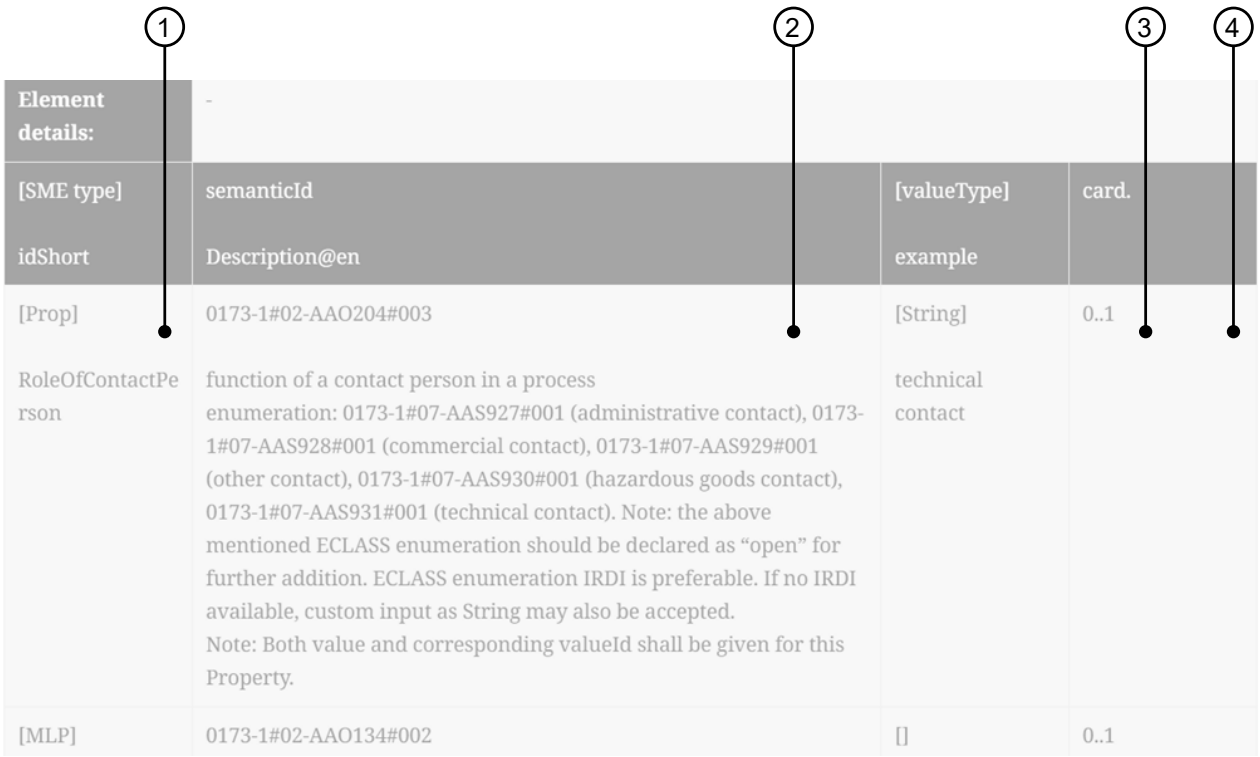

Figure 20 – Format of table row items

Figure 20 shows the format of the individual row items of such table. For the different row items, the following provisions are given by Table 8:

Table 8 – Format of table row items

ID	Label	Provision
1	[SME type] idShort	In the first line, the type of the AAS element shall be given. Abbreviation according [6] is allowed. The information shall be surrounded by '[ ]'. In the second line, the idShort of the AAS element shall be given. Could be an idShort with enumeration template such as "__000__" or "__00__" (see section 9.4).
2	semanticId = [idType]value Description@en	In the first line, the semanticId of the AAS element shall be given. In the second line, the english language of the description of the AAS element shall be given. The AASX files for the SMT may specify further languages. If used to define ConceptDescription, this information shall be taken over to the definition of the ConceptDescription, depending on its data specification template. In the following lines, more information can be given: A line ending with an '@' and ISO 639 language code in lowercase can be used to specify a further language for description/ definition. A line starting with "Note: " shall be considered as a note in the SMT document and is not taken over to description/ definition. A line starting with "Constraint: " shall state an textual constraint on the usage of the AAS element. It is not taken over to description/ definition.
3	[valueType] example	If the AAS element is a Property, the valueType shall be given in the first line. The information shall be surrounded by '[ ]'.

		If the AAS element is a MultiLanguageProperty, '[langString]' shall be given in the first line. In the next line, one or more example values can be given. String values might end with an '@' and ISO 639 language code in lowercase.
4	card.	In the first line, the cardinality of the AAS element shall be given. This can be [1], [0..1], [0..*], [1..*].

After the table header, before the first row items, two table rows shall be given featuring the respective labels defined in
Table 8, as illustrated by Figure 20.

8 TemplateId and semanticId of SMT

8.1 General

The meta model of the AssetAdministrationShell distinguishes between the templateId and the semanticId of a Submodel. The semanticId references a concept definition (either as a ConceptDescription as defined in the metamodel of the AAS or as a reference to an external dictionary entry like ECLASS). The templateId is the ID of the Submodel itself. It is used if a Submodel shall be referenced. The following holds:

- A Submodel template is a Submodel with kind = Template (Submodel/kind = Template)
- A Submodel template has an ID, called the templateId (Submodel/id)
- A Submodel template adds a semanticId to the Submodel (Submodel/semanticId). This semanticId is used as value for the semanticId of the Submodel Instances (Submodels with Submodel/kind = Instance)
- A Submodel, which is guided by a Submodel template, should denote this by adding the ID of the Submodel template (i.e. its Submodel/id) to Submodel/administration/templateId.

For example, see Figure 21 and Figure 22.

Submodel	
Referable:	
idShort:	exampleSMT
HasExtension:	
Identifiable:	
id:	https://admin-shell.io/idta-02058-1-0
id (Base64):	aHR0cHM6Ly9hZG1pbi1zaGVsbC5pby9pZHRhLTAyMDU4LTEtMA==
Kind (of model):	
kind:	Template
Semantic ID:	
semanticId:	(GlobalReference) https://admin-shell.io/smt/example-smt

Figure 21 – TemplateId and semanticId of an SMT

Submodel	
Referable:	
idShort:	exampleSM
HasExtension:	
Identifiable:	
id:	http://example.com/myExampleSM
id (Base64):	aHR0cDovL2V4YW1wbGUuY29tL215RXhhbXBsZVNN
administration:	
templateId:	https://admin-shell.io/idta-02058-1-0
Kind (of model):	
Semantic ID:	
semanticId:	(GlobalReference) https://admin-shell.io/smt/example-smt

Figure 22 – Submodel guided by SMT

9 Qualifers and attributes of SMT elements

9.1 General

According IEC 63278-1, a SMT element specifies the structure for a SubmodelElement. Especially, it specifies, to which concept repository entry for an information, relation or service or hierarchical structure (so called purpose) the SubmodelElement is related to. For the AASX of SMT, each SMT element is a SubmodelElement with kind = Template and the relation is done via the semanticId of the SubmodelElement.

To express the particular purpose of the structure, attributes shall specify:

- Optional & Cardinality
- Either-Or
- Example & Default & Initial Values (Here: Value used for Example Value)
- allowed ranges of properties (e.g. temperature in range -60° to +200° Celsius)
- regular expressions for naming (e.g. Document__00__) and for allowed values of properties
- required languages for multi-language properties
- user access mode (read-write or read-only)

These attributes might be specified by dedicated means of ConceptDescriptions (e.g. see the SAMM model [R9]) or by means of Qualifiers [6] (see below).

Furthermore, the AASX Package Explorer [R3] allows usage of the plugin AasxPluginGenericForms to assist users in creating Submodels based on SMTs. For this purpose, further attributes are provided here (see below).

9.2 Qualifiers controlling the structure

The following Qualifiers may be used, to allow a SMT element to control the generation of instantiated SubmodelElements (see Table 9). These Qualifiers are available as presets for the AASX Package Explorer [R3].

Table 9 – Qualifiers controlling the structure

Qualifier/type*[3] (grey = legacy)[4]	Qualifier/semanticId* Description	Qualifier/value[5]*
SMT/Cardinality Multiplicity	https://admin-shell.io/SubmodelTemplates/Cardinality/1/0 This Qualifier allows to specify how many SubmodelElement instances of this SMT element are allowed in the actual collection (hierarchy level of the Submodel). Note: All SubmodelElement instances need to have a unique idShort. For this, a template string can be given in the idShort of the SMT element (see section 9.4).	Allowed: One ZeroToOne ZeroToMany OneToMany

[3] For *, confer to the meta model description in [6]

[4] In grey, legacy Qualifier names are indicated

[5] Allowed values and/ or example values are given

SMT/EitherOr	https://admin-shell.io/SubmodelTemplates/EitherOr/1/0 The Qualifier value defines an id of an equivalence class. Only ids in the range [A-Za-z0-9] are allowed. If multiple SMT elements feature the same equivalence class, only one of these are allowed in the actual collection (hierarchy level of the Submodel).	Examples: 1 LOLLIPOP
SMT/InitialValue	https://admin-shell.io/SubmodelTemplates/InitialValue/1/0 Specifies the initial value of the SubmodelElement instance when it is created for the first time.	Example, e.g. for property "Working days per week": 5
SMT/DefaultValue	https://admin-shell.io/SubmodelTemplates/DefaultValue/1/0 Specifies the default value of the SubmodelElement instance. Often, this might designate a neutral, zero or empty value depending on the valueType of a SMT element.	Examples: 0 ""
SMT/ExampleValue	https://admin-shell.io/SubmodelTemplates/ExampleValue/1/0 Specifies an example value of the SubmodelElement instance, in order to allow the user to better understand the intention and possible values of a SubmodelElement instance. Note: Multiple examples can be given by delimiting them by '|' In case of a translatable string (langString) the example value shall be an English example string. Alternative (to be decided): add suffix like @en to string to denote language.	Examples: 42 "Hello" 3.1415
SMT/AllowedRange	https://admin-shell.io/SubmodelTemplates/AllowedRange/1/0 Specifies a set of allowed continuous numerical ranges. Note: Multiple ranges can be given by delimiting them by '|'. Note: A single range is defined by interval start and end, either including or excluding the given number. Note: Interval start and end are delimited by ','; '.' is the decimal point Note: '*' allows to enter the default value	Examples: [0,10] (0.0,9.9] *|[1,6] [2,3]|[6,7]

SMT/AllowedIdShort	https://admin-shell.io/SubmodelTemplates/AllowedIdShort/1/0 Specifies a regular expression validating the idShort of the created SubmodelElement instance. Note: The format shall conform to POSIX extended regular expressions.	Example: Title[\d{2,3}]
SMT/AllowedValue	https://admin-shell.io/SubmodelTemplates/AllowedValue/1/0 Specifies a regular expression validating the value of the created SubmodelElement instance in its string representation. Note: the format shall conform to POSIX extended regular expressions.	Example: (red|green|blue)
SMT/RequiredLang	https://admin-shell.io/SubmodelTemplates/RequiredLang/1/0 If the SMT element is a multi-language property (MLP), it specifies the required languages, which shall be given. Note: Multiple languages can be given by multiple Qualifiers. Note: Multiple languages can be given by delimiting them by '|' Note: languages are specified either by ISO 639-1 or ISO 639-2 codes.	Example: en|fr
SMT/AccessMode	https://admin-shell.io/SubmodelTemplates/AccessMode/1/0 Specifies the user access mode for SubmodelElement instance. When a Submodel is received from another party, if set to Read/Only, then the user shall not change the value	Allowed: Read/Write Read/Only

9.3 Qualifiers supporting the generic forms functionality

9.3.1 List of Qualifiers

The following Qualifiers are defined to allow a SMT element to control the presentation of forms. These Qualifiers are available as presets for the AASX Package Explorer.

Table 10 – Qualifiers supporting the generic forms functionality

Qualifier.name*	Qualifier.semanticId* Description	Qualifier.value*
FormTitle	https://admin-shell.io/SubmodelTemplates/FormTitle/1/0 Allows adding a speaking title to the edit field, which could give particular hints for filling out the value.	Please use these fields to describe your product
FormInfo	https://admin-shell.io/SubmodelTemplates/FormInfo/1/0 Allows adding a longer explanation to the edit field, which could give particular hints for filling out the value.	Name of the series or product type named by the manufacturer
FormUrl	https://admin-shell.io/SubmodelTemplates/FormUrl/1/0 Specifies a hypertext link to an external URL for further explanations of the SMT element. The link is offered to be activated by the user.	https://en.wikipedia.org/wiki/RGBA_color_model
SMT/Cardinality Multiplicity	https://admin-shell.io/SubmodelTemplates/Cardinality/1/0 https://admin-shell.io/SubmodelTemplates/Multiplicity/1/0 This Qualifier allows to specify, how many SubmodelElement instances of this SMT element are allowed in the actual collection (hierarchy level of the Submodel). Note: All SubmodelElement instances need to have an unique idShort. For this, a template string can be given in the idShort of the SMT element (see section 9.4) Note: Multiplicity is legacy, use of SMT/Cardinality is recommended.	One ZeroToOne ZeroToMany OneToMany
EditIdShort	https://admin-shell.io/SubmodelTemplates/EditIdShort/1/0 Specifies, if the user is able to edit the idShort of the SubmodelElement instance within the form's presentation. Note: For simple applications, a value of "False" is recommended. Note: If the designed form allows to create multiple collections, "True" could be an option for the collection.	True False
EditDescription	https://admin-shell.io/SubmodelTemplates/EditDescription/1/0	True False

	Specifies, if the user is able to edit the description of the SubmodelElement instance within the form's presentation. Note: For simple applications, a value of "False" is recommended. Note: If the designed form allows to create multiple collections, "True" could be an option fo the collection.	
FormChoices	https://admin-shell.io/SubmodelTemplates/FormChoices/1/0 Allows multiple choices of value; list of values separated by semicolon ';'.	Apples; Pies; Peanuts

9.4 Enumeration template for speaking idShort designations

The idShort of each SubmodelElement shall be unique in its namespace, e.g. the containing collection. At the same time, many users want the idShort to be a speaking name. In order to facilitate the automatic generation of speaking names, an addition such as "__000__" or "__00__" might be included in the idShort of an SMT element, which cardinality is other than "One". The number of zeros will correspond to the number of digits of a continuous numerical index for idShort.

Example: If a Submodel might describe multiple records, and a single record is described by multiple properties collected in a SubmodelElementCollection (SMC) with parent as Submodel, the application will be as described in Figure 23.

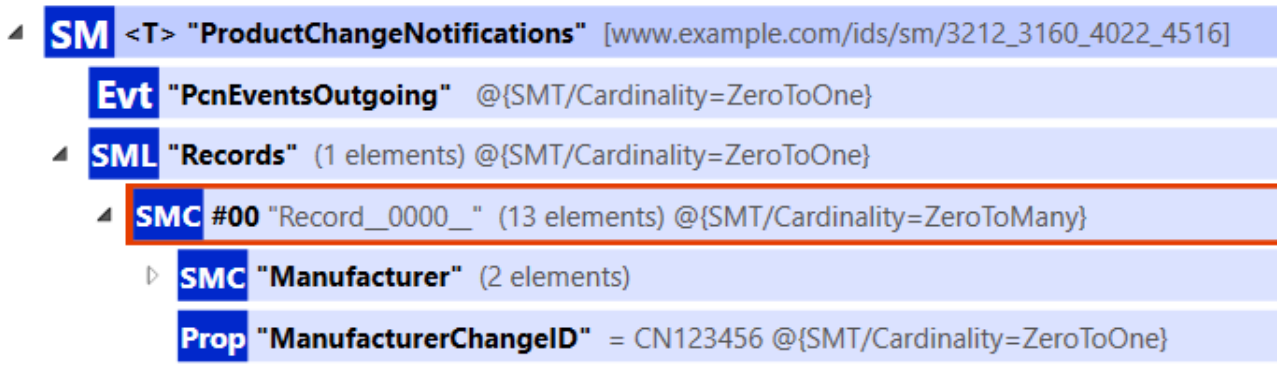

Figure 23 – Exemplary template idShort attribute for "Record"

Note: In version 3.0 of the meta model [6], SubmodelElementCollection is amended by SubmodelElementList. The idShort of such listed elements was forbidden.
In version 3.1 of the meta model, the use of idShort in such cases is allowed again.

9.5 Marking arbitrary content in SubmodelElement data

In certain cases, data of SubmodelElements is to be filled out by the user, which goes beyond strict specifications. This data cannot be left empty, because constraints of the meta model might be enforced by AAS frameworks (e.g. for version 3.0 of the meta model [6]). Therefore, the following values for References are foreseen:

Table 11 – Marking arbitrary content in SubmodelElement data

Marking	Reference and description
Arbitrary content	https://admin-shell.io/SMT/General/Arbitrary The idShort, description and semanticId of a SubmodelElement might be chosen arbitrarily, as described by the SMT specification.
Intentionally empty	https://admin-shell.io/SMT/General/IntentionallyEmpty No further details given by the SMT specification, however Reference set in order to comply with constraints.

9.6 SemanticIds and Extensions for the generation of AsciiDoc markup language

9.6.1 General

The use of SubmodelElements to generate AsciiDoc markup is described in section 3.8. A Submodel with dedicated SubmodelElements is created. For the generation of markup, SubmodelElements are processed in a linear manner (top to bottom, depth first). SubmodelElements with specific semanticIds (see section 9.6.2) are used to generate markup. Extensions on these SubmodelElements (see section 9.6.3) control the details of the generation.

As of today, the specifications in section 9.6.2 and section 9.6.3 are specific to the AsciiDoc generation of AASX Package Explorer [R3].

9.6.2 SemanticIds for the generation of AsciiDoc markup language

The following types of SubmodelElements with semanticIds are meaningful to the generation of AsciiDoc markup (see Table 12).

Table 12 – SubmodelElements and semanticIds for the generation of AsciiDoc markup language

Element	semanticId Description	Examples
Submodel	http://admin-shell.io/aasx-package-explorer/**functions/asciidoc**/1/0 Identifies the Submodel for generating AsciiDoc for the human-readable part of the SMT specification. The idShort of the Blob should point out, that AsciiDoc is generated and which SMT specification it refers to.	[idShort] AsciiDoc_DigiNameplate_2_0
Blob	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**heading1**/1/0 http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**heading2**/1/0 http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**heading3**/1/0	About this document [idShort] 1_1_About_this_document [contentType] text/markdown

	Defines a heading in the AsciiDoc. The three possible levels, given by three semanticIds, relate to chapters, sections and sub-sections of the document. Numbering is done automatically. The content of the Blob shall be set to the heading title; spaces and special characters are allowed. The idShort of the Blob should contain a speaking abbreviation of the heading respecting the constraints for idShorts (no spaces, only letters, digits, underscores and hyphens; hyphens and underscores not at the beginning und not at the end[6]). The contentType of the element shall be set to "text/markdown"	
Blob	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**textblock**/1/0 Defines one or multiple generic text blocks in the AsciiDoc. The content is treated as pure AsciiDoc markup and is taken 1:1 to the generated output. Multiple headings, paragraphs, tables, includes and more might be stated in the content. The idShort of the Blob should contain a speaking abbreviation of the content of the text block(s) respecting the constraints for idShorts (no spaces, only letters, digits, underscores, hyphens). The contentType of the element shall be set to "text/markdown"	List * A * B Table |======== | A1 | B1 | A2 | B2 |======== [idShort] Paragraph_Standards [contentType] text/markdown
Blob	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**coverpage**/1/0 Defines the starting point of the generation, describing important face matter of the document. For constraints of idShort, contentType see above.	see complete example [R13]
File	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**imagefile**/1/0 File element linking to a (supplemental) file and embedding link into the generated AsciiDoc markup. The File element shall link to an existing file. The contentType shall be given. The supplemental file will be copied to the generated files in the temporary working directory. Suitable files are binary files, such as images, but also e.g. large markup files. Encoding will be preserved. Via the Extension (see below), it can be controlled, if a reference/ link is written to the generated AsciiDoc markup.	/aasx/idta-smt-badge.png [contentType] image/png
Blob	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**imagefile**/1/0 Binary or text content, which shall be written as specific file to the generated files in the temporary working directory. The contentType shall be given. Via the Extension (see below), it can be controlled, if a reference/ link is written to the generated AsciiDoc markup.	ABC [contentType] text/markdown iVBORw0KGgoAAAANSUhEU…

[6] In V3.0 underscore at the end of an idShort is valid but this is not recommended.

	Note: .svg-format is recommended for vector content and .png-format is recommended for bitmap images.	[contentType] image/png
Reference-Element	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**generate-uml**/1/0 Identifies a structure of SubmodelElements which shall be generated as UML class diagram in PlantUML language. Via the Extension (see below), details of the generation process can be specified. The idShort is used to generate a dedicated PlantUML (.puml)-file for the input; the idShort shall be unique in the temporary working directory.	[Submodel] www.example.com/ids/sm/1225_9020_5022_1974 [SubmodelElementList] Markings
Reference-Element	http://admin-shell.io/aasx-package-explorer/functions/asciidoc/generate-tables/1/0 Identifies a structure of SubmodelElements which shall be generated as table in the generated markdown. Via the Extension (see below), details of the generation process can be specified.	[Submodel] www.example.com/ids/sm/1225_9020_5022_1974 [SubmodelElementList] Markings

9.6.3 Further semanticIds for identification of model contents

To allow maintaining multiple SMT models and respective AsciiDoc Submodels in a repository, some further semanticIds are required (see Table 13).

Table 13 – Further semanticIds for identification of model contents

semanticId / supplementalSemanticId **Description**	**Examples**
http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**target-model**/1/0 References from the AsciiDoc Submodel to the respective SMT model, which holds the template definition.	[SubModel] www.example.com/ids/sm/1225_9020_5022_1974
http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**section-scope**/1/0 Identifies the text block(s), which hold the scope statement of the SMT specification. This semanticId shall be attached as supplementalSemanticId. One or multiple SubmodelElements might by tagged with this semanticId. The heading shall not be tagged.	n/a
http://admin-shell.io/aasx-package-explorer/functions/asciidoc/**section-standards**/1/0 Identifies the text block(s), which hold the declaration of relevant standards to the SMT specification. This semanticId shall be attached as supplementalSemanticId. One or multiple SubmodelElements might by tagged with this semanticId. The heading shall not be tagged.	n/a

9.6.4 Extensions to control the generation of AsciiDoc markup language

The workflow with a single source AsciiDoc working draft document (see section 3.8) has the potential to significantly reduce manual efforts when compiling a SMT specification; however, this requires the necessity to tailor (generated) contents to the readers' needs. This tailoring of generation can be achieved by attaching Extensions to the respective SubmodelElements (see section 9.6.2) of the AsciiDoc markup generation.

The possible Extension attributes for the Extension name “ExportSmt.Args” are shown below (see Table 14). The different value attributes and value assignments can be combined by the known JSON rules.

Table 14 – Extensions to control the generation of AsciiDoc markup language

Extension **ExportSmt.Args** value attribute	Description	Example
noLink	For image file elements: If set to true, no link text in the produced AsciiDoc text will be created. File is simply created in temporary working directory.	{ "noLink" : "true" }
fileName	For image link elements: If set, the filename of the target file which is to be created. If not set, the filename will be derived (including extension) from the idShort.	{ " fileName " : "xyz.css" }
depth	For generate UML elements: controls, how many recursion levels in the class diagram will be generated. A level of 1 generates just one class given by the SubmodelElements given by the ReferenceElement. A level of 2 generates also the direct child classes of the referenced class. The default is to generate all child classes. This could lead to very large UML class diagrams.	{ "depth" : 2 }
width	If set, it will constrain the element (image) to a certain width in percent in a range of [1..100].	{ "width" : 50 }
uml	Specifies more options for the UML generation in a separate data structure. Therefore, the following value attributes are stated with a preceding "uml.".	{ "uml" : { "Outline": true, "SwapDirection": true } }
uml.Suppress	This attribute will suppress complete (child) classes in the UML class diagram to improve clarity of presentation. Strings delimited by spaces will be matched against class names and on positive match will suppress rendering of such class.	{ "uml" : { "Suppress" : "Contact" } }
uml.LimitInitialValue	This attribute might help to keep class diagrams geometrically compact. If greater or equal zero, limits the number of characters for initial values in the class members list.	{ "uml" : { " LimitInitialValue " : 15 } }
uml.Outline	This attribute causes classes to show no individual members. By reducing this level of detail, multi-level class diagrams can be used to e.g. to give a complete overview of the Submodel template.	{ "uml" : { "Outline": true } }

uml.SwapDirection	If set, changes the direction of adding graphical elements with the PlantUML language. Class diagrams might be easily to read.	{ "uml" : { "SwapDirection": true } }

10 ConceptDescriptions for SMT

10.1 General

For a Submodel template, each Submodel element template is semantically defined by referring to a semanticId, which identifies the semantic definition of that element.

10.2 Use of existing concept repository items

The meta model of the AAS and the AASX PackageExplorer [R3] allows expressing ConceptDescriptions conformant to IEC 61360, e.g., for Properties. Describing such ConceptDescriptions e.g., enables defining units of measure and is therefore important for practical application. The semanticId of a Submodel element template refers to the identifier of such ConceptDescription.

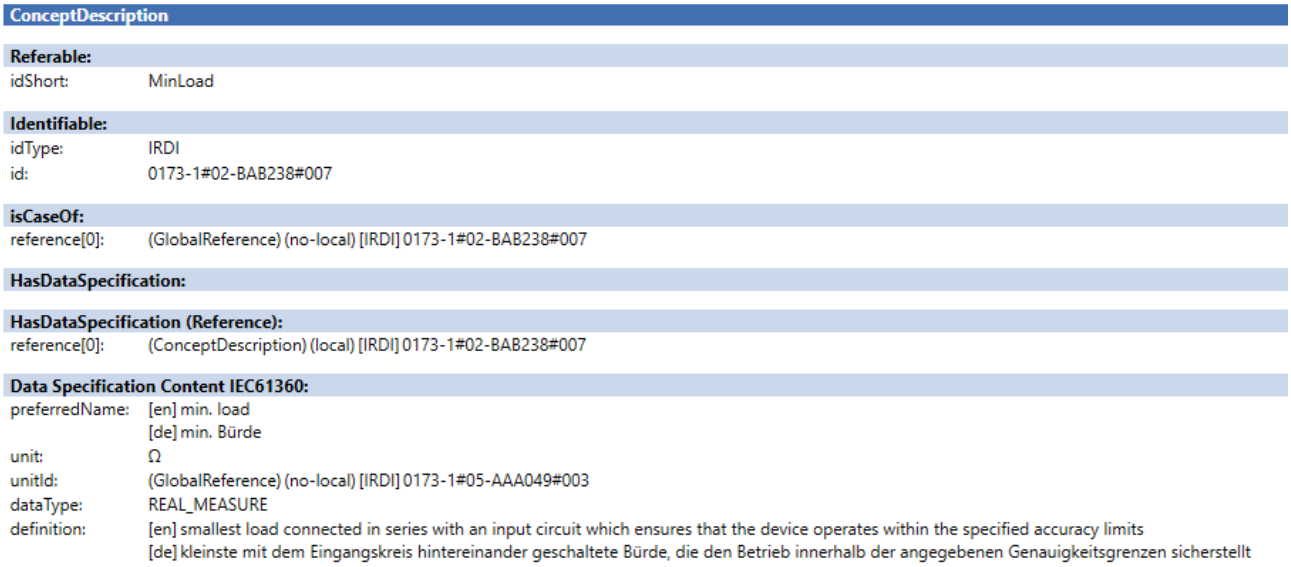

Figure 24 – Example of a ConceptDescription in AASX Package Explorer [R3]

The delivery of the SMT shall include all definitions of ConceptDescriptions which relate directly to the usage of the respective SMT. In particular, the ConceptDescriptions shall include definitions used in the definition tables described in 7.3.

Note: Please be aware that dictionaries above do not provide automatic retrieval (e.g. by REST interface); the automatic provision of units of measure is not possible unless ConceptDescriptions are provided within the AASX package.

10.3 Description of new concept repository items

If a concept is not already described by publicly accessible dictionaries, such as ECLASS, IEC CDD or [R4], a new concept repository item needs to be created, that is, a new identifier needs to be reserved for such a concept repository item and the respective ConceptDescription. This can be achieved by starting a standardization process in concept repositories such as ECLASS or IEC CDD. Alternatively, an identifier can be reserved utilizing [R4]. The main site of [R4] describes the registration process for such identifiers.

Note: By registering an identifier in [R4], an identifier is uniquely assigned to the ConceptDescription. This does not necessarily mean that a URL resource (e.g. HTML page) needs to exist 'behind' this identifier. The pure existence of the identifier is sufficient.

Annex A. Explanations on used table formats

Note: This annex shall be included in each SMT document (see section 7.1) and is included here for illustration.

1. General

The tables used in this document try to outline information as concisely as possible. They do not convey all information on Submodels and SubmodelElements. For this purpose, the definitive definitions are given by a separate file in form of an AASX file of the SMT and its elements.

2. Tables on Submodels and SubmodelElements

For clarity and brevity, a set of rules is used for the tables for describing Submodels and SubmodelElements.

- The tables follow in principle the same conventions as in [5].
- The table heads abbreviate 'cardinality' with 'card'.
- The tables often place two information elements in different rows of the same table cell. In this case, the first information is marked out by sharp brackets [] form the second information. A special case are the semanticIds, which are marked out by the format: (type)(local[7])[idType]value.
- The types of SubmodelElements are abbreviated (among others):

Table 15 – Abbreviations of SubmodelElements

SME type	SubmodelElement type
Blob	Blob
Cap	Capability
Ent	Entity
Evt	Event
File	File
MLP	MultiLanguageProperty
Opr	Operation
Prop	Property
Range	Range
Ref	ReferenceElement
Rel	RelationshipElement
RelA	AnnotatedRelationshipElement
SMC	SubmodelElementCollection
SML	SubmodelElementList

- If an idShort ends with '__00__', this indicates a suffix of the respective length (here: 2) of decimal digits, in order to make the idShort unique. A different idShort might be chosen, as long as it is unique in the parent's context.
- The Keys of semanticId in the main section feature only idType and value, such as: [IRI]https://admin-shell.io/vdi/2770/1/0/DocumentId/Id, as described in the serialization of the AAS meta model. The attributes "type" and "local" (typically "ConceptDescription" and "(local)" or "GlobalReference" and (no-local)") need to be set accordingly; see [6].
- If a table does not contain a column with "parent" heading, all represented attributes share the same parent. This parent is denoted in the head of the table.

[7] "Attribute "local" was removed in V3.0RC01 [6]

- Multi-language strings are represented by the text value, followed by '@'-character and the ISO 639 language code: example@EN.
- The [valueType] is only given for Properties.

Annex B. Resources

The following resources are available to help develop SMT specifications. Resources with heading backslash refer to the IDTA Teams drive.

Table 16 – Resources used in the document

Ressource	Description
[R1]	Process Description. Registration of Asset Administration Shell Submodel templates for Digital Twins. Industrial Digital Twin Association (IDTA). V1.0 https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2022/01/2021-12-01_IDTA_Process-Submodels_V1.0.pdf
[R2]	IDTA comments template \IDTA - WG2 - Submodels\Submodel-Development\00_Templates\03 IDTA Submodel Spec Comments Template.xlsx
[R3]	AASX Package Explorer https://github.com/eclipse-aaspe/aaspe/releases Previous releases can be found here: https://github.com/admin-shell-io/aasx-package-explorer/releases
[R4]	IDTA GitHub Asset Administration Shell Identifiers https://github.com/admin-shell-io/id
[R5]	Help file for [R3] to import/ export tables https://github.com/eclipse-aaspe/aaspe/tree/main/src/AasxPluginExportTable/help
[R6]	Predefined concepts for AASX Package Explorer https://github.com/eclipse-aaspe/aaspe/tree/main/src/AasxPredefinedConcepts
[R7]	PlantUML https://plantuml.io
[R8]	ISO/IEC 19505-2:2012, Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure
[R9]	Semantic Aspect Meta Model (SAMM) https://eclipse-esmf.github.io/samm-specification/snapshot/index.html https://projects.eclipse.org/projects/dt.esmf
[R10]	Homepage: “AsciiDoc”. https://asciidoc.org/
[R11]	IDTA GitHub repository for Submodel templates. https://github.com/admin-shell-io/submodel-templates
[R12]	AsciiDoc tools: https://docs.asciidoctor.org/ https://pandoc.org/ https://docs.asciidoctor.org/asciidoctor/latest/migrate/ms-word/ https://www.vertopal.com/en/convert/docx-to-asciidoc

	https://tableconvert.com/asciidoc-generator
[R13]	AsciiDoc templates, header & style files for SMT specifications. https://industrialdigitaltwin.sharepoint.com/:f:/r/sites/IDTA-SubmodelsPrivateTrack/Freigegebene%20Dokumente/Workstream%20%E2%80%9EProcess%E2%80%9C/02%20Best%20practice%20How%20to%20write%20a%20SMT/AsciiDoc_Template_files?csf=1&web=1&e=oPW0cy Convert Word to AsciiDoc: https://github.com/admin-shell-io/word2asciidoc
[R14]	Command lines for docker containers for rendering AsciiDoc to HTML / PDF (using some of the files in [R13]): For HTML: **docker run -it -v .:/documents/ asciidoctor/docker-asciidoctor asciidoctor -r asciidoctor-diagram -a toc=left -a stylesheet=asciidoc-style-idta.css *.adoc** For PDF with line numbers: **docker run -it -v .:/documents/ asciidoctor/docker-asciidoctor asciidoctor-pdf -r asciidoctor-diagram -r ./extended.rb -a toc=macro -a pdf-theme=my-theme.yml -a env-pdf *.adoc**

Annex C. Change log

Major changes compared to version 1.0 of this document:

- Artifacts to be delivered (see 3.2)
- Added two workflows for AsciiDoc (see section 3)
- Definition of SMT Dropins (see section 4)
- Adapted table formats to AsciiDoc (see section 7)
- Clarify templateId and semanticId (see section 8)
- Marking arbitrary content in SubmodelElement data (see 9.5)
- SemanticIds and Extensions for the generation of AsciiDoc markup language (see 9.6)

Annex D. Bibliography

[1] "Recommendations for implementing the strategic initiative INDUSTRIE 4.0", acatech, April 2013. [Online]. Available: Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group - acatech - National Academy of Science and Engineering

[2] "Implementation Strategy Industrie 4.0: Report on the results of the Industrie 4.0 Platform"; BITKOM e.V. / VDMA e.V., /ZVEI e.V., April 2015. [Online]. Available: Implementation Strategy Industrie 4.0 (bitkom.org)

[3] "The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany", March 2018, [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.html

[4] "Beispiele zur Verwaltungsschale der Industrie 4.0-Komponente – Basisteil (German)"; ZVEI e.V., Whitepaper, November 2016. [Online]. Available: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/November/Beispiele_zur_Verwaltungsschale_der_Industrie_4.0-Komponente_-_Basisteil/Beispiele-Verwaltungsschale-Industrie-40-Komponente-White-Paper-Final.pdf

[5] "Verwaltungsschale in der Praxis. Wie definiere ich Teilmodelle, beispielhafte Teilmodelle und Interaktion zwischen Verwaltungsschalen (in German)", Version 1.0, April 2019, Plattform Industrie 4.0 in Kooperation mit VDE GMA Fachausschuss 7.20, Federal Ministry for Economic Affairs and Energy (BMWi), Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2019-verwaltungsschale-in-der-praxis.html

[6] IDTA-01001-3-0 "Specification of the Asset Administration Shell. Part 1: Metamodel", V3.0, April 2023, [Online]. Available: https://industrialdigitaltwin.org/en/content-hub/aasspecifications

[7] "IEC 63278-1 ED1: Asset Administration Shell for industrial applications – Part 1: Asset Administration Shell structure"

[8] "IDTA 02006-2-0 Digital Nameplate for industrial equipment", Version 2.0, 20. October 2022, Available: https://github.com/admin-shell-io/submodel-templates/tree/main/published/Digital%20nameplate/2/0

www.industrialdigitaltwin.org