IDTA 02027 Asset Interfaces Mapping Configuration

Version 2.0

May 2026

SPECIFICATION

Submodel Template
IDTA approved
- 100% AAS compliant
- Consistent & interoperable
- Released by the AAS experts

Submodel Template of the
Asset Administration Shell

IDTA 02027

Imprint

Publisher

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

Version history

Date	Version	Comment
21.05.2026	2.0	Release of the official Submodel template published by IDTA.

Table of Contents

Chapter 1. General

1.1. About this Document

This document is a part of a specification series. Each part specifies the contents of a Submodel template for the Asset Administration Shell (AAS). The AAS is described in [1], [2], [3] and [6]. First exemplary Submodel contents were described in [4], while the actual format of this document was derived from the "Administration Shell in Practice" [5]. The format aims to be very concise, giving only minimal necessary information for applying a Submodel template, while leaving deeper descriptions and specification of concepts, structures and mapping to the respective documents [1] to [6].

The target group of the specification are developers and editors of technical documentation and manufacturer information, who are describing assets in smart manufacturing by means of the Asset Administration Shell (AAS) and therefore need to create a Submodel instance with a hierarchy of SubmodelElements. This document especially details on the question which SubmodelElements with which semantic identification shall be used for this purpose.

1.2. Scope of the Submodel Template

The Asset Interfaces Mapping Configuration (AIMC) Submodel version 2.0 supports two use cases:

1. Asset-to-AAS mapping
   Together with the Asset Interface Description (AID) Submodel, AIMC specifies how data provided by assets or asset-related services is mapped to target locations in an Asset Administration Shell (AAS). In this scenario, AIMC functions as a configuration Submodel for northbound communication between an asset and the enclosing AAS (for example, an application-specific Submodel for monitoring or logging).
2. AAS-to-AAS mapping
   The AIMC can also specify mappings between locations within an AAS. For example, compute a singular value from multiple existing elements.

In both cases, a mapping defines the relationship between a set of sources (input data) and a set of sinks (output data) where results are written. An optional transformation step may be applied between sources and sinks to convert or process the input data into the desired output format or value.

In conjunction with the AID, the AIMC can help meet regulatory compliance requirements, including those under the EU Digital Product Passport and the EU Data Act. It achieves this by exposing machine-related data via the AAS in a standardized, machine-readable form. Even though the AID and AIMC are closely related, they serve different purposes and are therefore separate Submodels. The AID documents which interfaces an asset offers and is typically provided by the asset provider. The AIMC, by contrast, defines how data is mapped in one direction, optionally including a transformation step. This can be a mapping between an asset and an AAS by leveraging the AID, or a mapping between two AASs (AAS A to AAS B) without involving the AID. AIMC configurations are typically created by the asset user. Finally, the AIMC and AID do not have to be co-located in the same AAS and may be hosted separately.

Figure 1 illustrates how the two AIMC use cases can be combined into a two-step workflow. First, data is onboarded into the AAS by using the AID and AIMC together. Once the data is available in the AAS, the AIMC can be used independently of the AID to define how existing AAS data is mapped and optionally transformed into higher-level signals, such as KPIs, and written to other locations within the AAS domain.

Example AIMC usage patterns:

- Example 1: Onboard asset data into its own AAS. The AAS shall be populated with data coming directly from the asset it represents. Consequently, the AID and AIMC are both integrated into the AAS of the asset to onboard asset data into the AAS.

- Example 2: Onboard machine production data into a product AAS. The AAS of a product shall integrate product-specific production data from a machine that manufactures the product. The AIMC mapping capabilities are used to describe how selected machine data, identified via the machine's AID, are transformed into the product AAS.
- Example 3: Aggregate device state data into a production line AAS. The AAS shall aggregate signals from multiple devices of a production line to derive an overall "machine is running" signal. Each device provides live state data via its own AAS. The production line AAS is extended with an AIMC that maps the relevant device-state sources and uses its optional transformation step to compute the aggregated running signal, which is then written to a defined location in the production line AAS.

The AIMC 2.0 adapts to the structure and protocols supported by the AID 1.1 standard: Modbus, HTTP, MQTT, OPC UA, BACnet. Subsequent releases of AID versions may trigger the release of adapted AIMC versions.

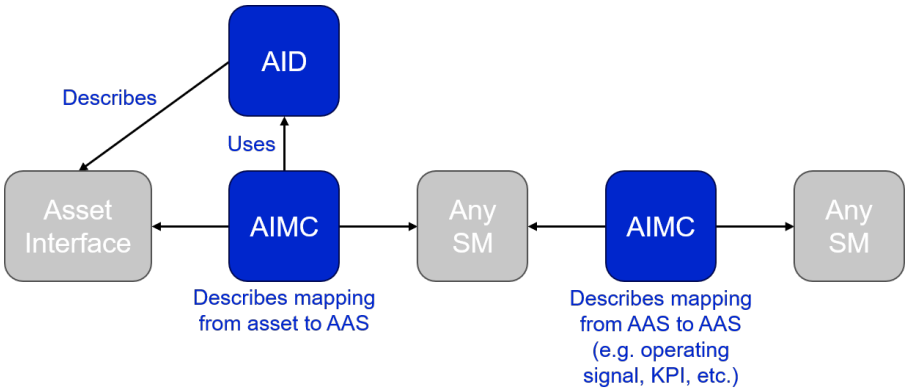

Figure 1. AIMC Use Cases

1.3. Data Mapping Processor

The Data Mapping Processor (DMP) is a software component that operationalizes the AIMC. It establishes and maintains the data flows defined by one or more AIMC Submodels, uses the referenced AIDs to connect to asset interfaces, executes optional transformations, and writes the resulting values to the configured target locations in the AAS. The DMP is typically implemented and operated by the AAS platform/service provider, or by an independent integration service deployed alongside the AAS. While the AIMC content is authored by the asset user or system integrator, the DMP provides the runtime that executes these configurations. The implementation details of the DMP are beyond the scope of this document but its main functionality is outlined here.

Figure 2 illustrates an exemplary scenario with two assets: (i) a data store with a REST interface and (ii) a motor publishing data via MQTT. Each asset provides an AID Submodel describing how its interface can be accessed. In addition, for each asset an AIMC Submodel is defined that specifies which data of the corresponding asset is collected, how it is transformed, and to which Submodel location the results are written. A target location can be any addressable Submodel element, for example a Property in a monitoring Submodel that stores the motor's voltage.

The numbered elements in Figure 2 can be interpreted as follows:

1. The DMP is a piece of software that establishes the data flow and data transformations as defined by the linked AIMC and AID Submodels.
2. Each linked AIMC consists of a list of mapping configurations. Each mapping configuration details the needed input data (coming from assets defined by AIDs or from other Submodels via REST [11]), how that data is transformed and where the results are written. A target location, where data is written to, can be any Submodel location.
3. The linked AIDs provide the configuration required to establish communication with their respective asset interfaces, such as the data stream of a temperature sensor.

4. Each *MappingConfiguration*, originating from its respective AIMC, defines N data inputs, an optional transformation of these inputs in a Lua script [18] and M outputs where the results of the transformation are written to. For synchronous protocols, such as REST, the DMP retrieves input values based on a configured polling interval. Therefore, at least one polling interval, either the *DefaultPollingInterval* or the local *PollingInterval*, must be defined for such sources. For asynchronous protocols, such as MQTT, inputs are delivered to the DMP as updates occur. Whenever any input value of a *MappingConfiguration* changes, the DMP re-evaluates the mapping (including transformation, if present) and propagates the updated result(s) to the configured output locations.

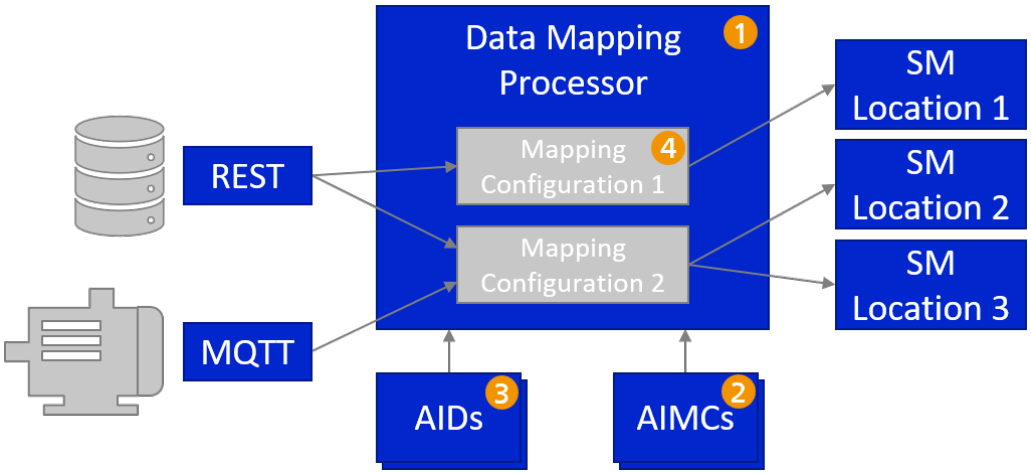

Figure 2. Overview of the Data Mapping Processor

1.4. Data Transformations

Many mapping scenarios require more than a direct one-to-one assignment of values, for example unit conversions, scaling, thresholding, value normalization, or the computation of higher-level signals (e.g., KPIs). For this purpose, the AIMC allows an optional transformation step between a set of input sources and one or more output sinks. This section specifies how such transformations are defined and executed in Lua (see Annex B.1 for the rationale behind Lua).

```
function aimc_main(sources)
  local value_1 = sources["SourceId1"]
  local value_2 = sources["SourceId2"]

  return {
    SinkId1 = value_1 + value_2,
    SinkId2 = math.sqrt(value_1*value_1 + value_2*value_2)
  }

  --[[ Alternative if a SinkId is not a valid Lua identifier.
  return {
    ["Sink Id1"] = value_1 + value_2,
    ["Sink-Id2"] = math.sqrt(value_1*value_1 + value_2*value_2)
  }--]]
end
```

The Lua script above defines the function *aimc_main*, which shall be present in every Lua transformation and serves as the entry point for execution. For each *MappingConfiguration*, the DMP passes all configured inputs from the *Sources* list to *aimc_main* via the *sources* table. Input values can be accessed by their respective *SourceId*.

Within *aimc_main*, the input values may be processed using standard Lua functionality, for example to perform scaling, unit conversion, validation, or to compute derived metrics such as a machine's OEE. The function is expected to return a table that assigns an output value to each sink. This return table links every *SinkId* defined in the *MappingConfiguration* to its corresponding value.

The Lua script is stored in the *MappingConfiguration's Transformation* element as a Blob with contentType set to "text/plain". Authors may define additional helper functions to structure more complex logic, but the transformation should remain compact. To preserve comprehensibility and maintainability, each *MappingConfiguration* is recommended to perform only one coherent logical task.

The *Transformation* Blob is optional. If no transformation is provided, only simple one-to-one data forwardings from source to sink are possible without any transformation logic in between. In this case, the number of sources and sinks shall be equal, and each source shall map to exactly one sink. To establish the relationship for each source-sink pair, they must have the same ID value (*SourceId* and *SinkId*, respectively) assigned.

In AIMC version 2.0, Lua transformations shall be stateless and are not allowed to persist state across executions (e.g., by using global variables). This restriction may be revisited in future AIMC versions.

A prototypical example demonstrating the dynamic execution of a Lua transformation in Python is provided in Annex B.2.

1.5. Relevant Standards for the Submodel Template

- W3C Web of Things Thing Description [7]
- Modbus [8]
- MQTT [9]
- HTTP [10]
- OPC UA [12]
- BACnet [13]
- IO-Link [14-16]
- IDTA Submodel Template: Asset Interfaces Description [17]
- IDTA Submodel Template: Process Variables for Manufacturing Key Performance Indicators [19]

1.6. Use Cases

This section describes exemplary use cases of the AIMC. It is assumed that component suppliers deliver AASs of their assets including AID Submodels that describe the interfaces of each asset. In this context, the AAS of an asset has to be hosted somewhere to allow for programmatic access by the user. Users of such assets can then leverage those AID Submodels to accelerate asset integration into their I4.0 ecosystem. The precise AAS location where asset data is written to and how that data is transformed along the way, has to be defined by the user in a dedicated AIMC Submodel.

It is assumed that the use-case implementers have software (e.g. the DMP) available for deploying the data pipelines laid out in the respective AIMC Submodels.

Table 1. AIMC Use Cases

Use Cases	Explanation
Asset Data Integration	A systems integrator (SI) wants to onboard a new device (e.g. a sensor) so that the data flows into the AAS ecosystem of the SI. The following workflow describes the necessary steps to establish the data integration of the device: 1. The SI connects to the AAS of the device. 2. The SI inspects the AID Submodel describing which data the device provides through which interfaces and with which security requirements. 3. The SI selects relevant data points from the relevant device interfaces. 4. The SI creates an AAS sink such as a separate monitoring, logging or OperationalData Submodel where the device data shall be written to. This Submodel can for instance be part of the AAS of the device. 5. The SI creates the AIMC Submodel that connects the selected device data points with the respective Submodel locations in the sink. 6. The SI deploys the AIMC Submodel along with the needed security requirements to activate the configured data pipelines.
Data Transformation (Simple)	The integrator of a temperature sensor wants to convert the measurement data from degrees Fahrenheit to degrees Celsius and integrate the converted value into the AAS of the sensor. The temperature sensor publishes its measurements via OPC UA. The following workflow describes the necessary steps to establish the data conversion of the sensor: 1. The integrator identifies the relevant data point in the AID Submodel of the sensor. 2. The integrator creates an OperationalData Submodel within the AAS of the sensor. 3. The integrator creates an AIMC Submodel together with a MappingConfiguration within the AAS of the sensor and adds the temperature value of the OPC UA interface coming from the AID as source. As sink serves a temperature Property located in the OperationalData Submodel. 4. The integrator implements the temperature conversion in Lua according to the definition in Section 1.4 and adds it to the Transformation Blob of the MappingConfiguration. 5. The integrator deploys the AIMC Submodel along with the needed security requirements to establish the data flow.

Data Transformation (Complex)	An automation engineer is tasked with calculating the Overall Equipment Effectiveness (OEE) of a machine. During onboarding of the machine in the Manufacturing Execution System (MES), the Process Variables for Manufacturing Key Performance Indicators (PVM) Submodel [19] was used. This Submodel contains all relevant variables for calculating the OEE, according to ISO 22400-2, including current machine data. The company uses a company-specific Submodel (custom) to record the OEE values. This Submodel contains four OEE-related Properties: Availability, Performance, Quality and OEE. The latter is a multiplication of the first three. The following workflow describes the necessary steps to convert PVM Submodel data to the company-specific OEE Submodel: 1. The engineer locates the machine's AAS in the local AAS repository. 2. The engineer finds the PVM Submodel in the machine's AAS. 3. The engineer adds an empty instance of the company-specific OEE Submodel to the machine's AAS. 4. The engineer creates an AIMC Submodel. For each OEE-related Property, the engineer adds a MappingConfiguration by taking the following steps: a. The engineer adds the sink relation by referencing the particular OEE-related Property in the Submodel. b. The engineer adds all required sources, based on the ISO 22400-2 specification, from the PVM Submodel by referencing the appropriate PVM Properties. c. The engineer implements the data transformation by writing a Lua script that implements the ISO 22400-2 prescribed calculation for the particular OEE-related Property. Note 1: The sub-steps above may be taken in any arbitrary order. Note 2: The engineer may also choose to use one MappingConfiguration with multiple sources and sinks to accomplish the full task. 5. The engineer deploys the AIMC Submodel along with the needed security requirements to activate the configured data pipelines. 6. The data is now flowing from PVM Submodel to the OEE Submodel.
Data Writing (Upcoming)	A factory operator wants setpoints to be written from the MES directly to a machine. For that, the AID exposes the interface of the MES and the machine which can then be used by the AIMC to define the data mappings between both systems. This is an upcoming use case since the AID 1.1 currently does not support writing data to assets.

Chapter 2. Submodel AIMC

2.1. Approach

The Submodel consists of an AIMC core part (see Figure 3) that specifies the basic structure of the AIMC Submodel.

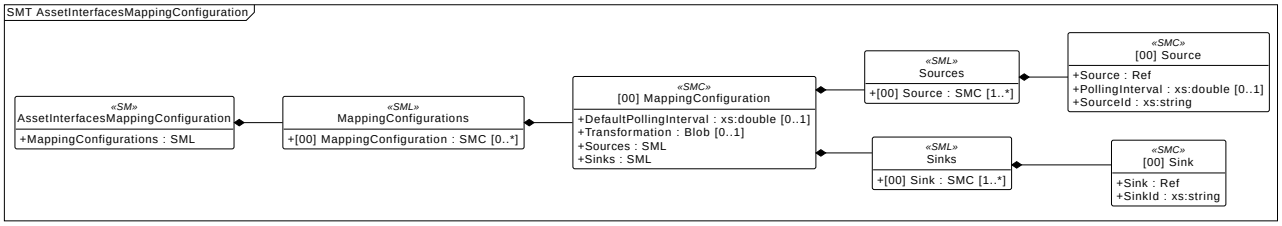

Figure 3. Elements of the SM AssetInterfacesMappingConfiguration

2.2. Overview of the AIMC Core Structure

The AIMC Submodel configures the mapping of data coming from interfaces (sources) to Submodel target locations (sinks) in form of SMCs provided in the *MappingConfigurations* SML. In each *MappingConfiguration* SMC, information is provided about needed sources, their transformation and sinks where the results are written to. Each source and sink is specified as reference element of the type *ModelReference* that points to the SubmodelElement of interest respectively. Consequently, sources and sinks can also be referenced from external AASes and do not have to be part of the same AAS where the AIMC is held.

2.3. Elements of the SM “AssetInterfacesMappingConfiguration“

For the Submodel, these important attributes need to be set:

Table 2. Elements of the SM AssetInterfacesMappingConfiguration

idShort:	**AssetInterfacesMappingConfiguration**		
Class:	Submodel		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/Submodel		
Parent:	-		
Explanation:	The AIMC 2.0 is used to describe how data is mapped from asset to AAS or from AAS to AAS.		
Element details:	-		
[SME type] idShort	semanticId Description@en	[valueType] example	card.

[SML] MappingConfigurations	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/1/0/MappingConfigurations List of MappingConfigurations that each map and transform data from their sources to their sinks. List of MappingConfigurations that each map and transform data from their sources to their sinks.	[] 1 elements	1

2.4. Elements of SML “MappingConfigurations”

Table 3. Elements of SML MappingConfigurations

idShort:	**MappingConfigurations**		
Class:	SubmodelElementList		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/1/0/MappingConfigurations		
Parent:	AssetInterfacesMappingConfiguration		
Explanation:	List of MappingConfigurations that each map and transform data from their sources to their sinks.		
Element details:	orderRelevant=No, semanticIdListElement=[GlobalReference, https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration], typeValueListElement=SubmodelElementCollection		
[SME type] idShort	semanticId Description@en	[valueType] example	card.
[SMC] MappingConfiguration	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration A MappingConfiguration defines one logical unit of sources (inputs) and sinks (outputs) that are in relation to one another. The relation can be expressed via a transformation. A MappingConfiguration defines one logical unit of sources (inputs) and sinks (outputs) that are in relation to one another. The relation can be expressed via a transformation.	[] 4 elements	0..*

2.5. Elements of SMC “MappingConfiguration”

Table 4. Elements of SMC MappingConfiguration

idShort:	**MappingConfiguration**
Class:	SubmodelElementCollection
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration
Parent:	MappingConfigurations
Explanation:	A MappingConfiguration defines one logical unit of sources (inputs) and sinks (outputs) that are in relation to one another. The relation can be expressed via a transformation.

Element details:	-		
[SME type] idShort	semanticId Description@en	[valueType] example	card.
[Prop] DefaultPollingInterval	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/DefaultPollingInterval The DefaultPollingInterval defines the default time interval in seconds for fetching new data from the synchronous data sources defined in this MappingConfiguration. It must be greater than zero for synchronous protocols (e.g. HTTP) that need polling and is ignored for asynchronous protocols (e.g. MQTT). The DefaultPollingInterval defines the default time interval in seconds for fetching new data from the synchronous data sources defined in this MappingConfiguration. It must be greater than zero for synchronous protocols (e.g. HTTP) that need polling and is ignored for asynchronous protocols (e.g. MQTT).	[Double]	0..1
[Blob] Transformation	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Transformation The transformation allows for transforming incoming data before writing it to the sinks. The transformation must contain an "aimc_main(sources)" entrypoint function in Lua. The transformation allows for transforming incoming data before writing it to the sinks. The transformation must contain an "aimc_main(sources)" entrypoint function in Lua.	[] 0 bytes	0..1
[SML] Sources	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sources This list includes all data sources that are used in this MappingConfiguration. This list includes all data sources that are used in this MappingConfiguration.	[] 1 elements	1
[SML] Sinks	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sinks This list includes all data sinks that are used in this MappingConfiguration. This list includes all data sinks that are used in this MappingConfiguration.	[] 1 elements	1

2.6. Elements of SML “Sources”

Table 5. Elements of SML Sources

idShort:	**Sources**		
Class:	SubmodelElementList		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sources		
Parent:	MappingConfiguration		
Explanation:	This list includes all data sources that are used in this MappingConfiguration.		
Element details:	orderRelevant=No, semanticIdListElement=[GlobalReference, https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source], typeValueListElement=SubmodelElementCollection		
[SME type] idShort	semanticId Description@en	[valueType] example	card.
[SMC] Source	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source A data source is defined by a Source reference, PollingInterval and SourceId. A data source is defined by a Source reference, PollingInterval and SourceId.	[] 3 elements	1..*

2.7. Elements of SMC “Source”

Table 6. Elements of SMC Source

idShort:	**Source**		
Class:	SubmodelElementCollection		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source		
Parent:	Sources		
Explanation:	A data source is defined by a Source reference, PollingInterval and SourceId.		
Element details:	-		
[SME type] idShort	semanticId Description@en	[valueType] example	card.

[Ref] Source	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source/Source This holds a reference to the respective SubmodelElement used as data source. A data source can be any SubmodelElement including those defined in the InteractionMetadata of AID Submodels for fetching live-data from assets. This holds a reference to the respective SubmodelElement used as data source. A data source can be any SubmodelElement including those defined in the InteractionMetadata of AID Submodels for fetching live-data from assets.	[]	1
[Prop] PollingInterval	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source/PollingInterval The PollingInterval defines the time interval in seconds for fetching new data from the given synchronous data source. It must be greater than zero for synchronous protocols (e.g. HTTP) that need polling and is ignored for asynchronous protocols (e.g. MQTT). It overwrites the DefaultPollingInterval of the respective MappingConfiguration of this source. The PollingInterval defines the time interval in seconds for fetching new data from the given synchronous data source. It must be greater than zero for synchronous protocols (e.g. HTTP) that need polling and is ignored for asynchronous protocols (e.g. MQTT). It overwrites the DefaultPollingInterval of the respective MappingConfiguration of this source.	[Double]	0..1
[Prop] SourceId	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Source/SourceId This is a unique and non-empty identifier that facilitates data access in the Lua transformation or establishes a relationship to a corresponding sink when no transformation is given. It must only be unique with respect to the Sources-list of the parent MappingConfiguration and not globally. This is a unique and non-empty identifier that facilitates data access in the Lua transformation or establishes a relationship to a corresponding sink when no transformation is given. It must only be unique with respect to the Sources-list of the parent MappingConfiguration and not globally.	[String]	1

2.8. Elements of SML “Sinks”

Table 7. Elements of SML Sinks

idShort:	**Sinks**

Class:	SubmodelElementList		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sinks		
Parent:	MappingConfiguration		
Explanation:	This list includes all data sinks that are used in this MappingConfiguration.		
Element details:	orderRelevant=No, semanticIdListElement=[GlobalReference, https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sink], typeValueListElement=SubmodelElementCollection		
[SME type] idShort	semanticId Description@en	[valueType] example	card.
[SMC] Sink	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sink A data sink is defined by a Sink reference and SinkId. A data sink is defined by a Sink reference and SinkId.	[] 2 elements	1..*

2.9. Elements of SMC “Sink”

Table 8. Elements of SMC Sink

idShort:	**Sink**		
Class:	SubmodelElementCollection		
semanticId:	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sink		
Parent:	Sinks		
Explanation:	A data sink is defined by a Sink reference and SinkId.		
Element details:	-		
[SME type] idShort	semanticId Description@en	[valueType] example	card.
[Ref] Sink	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sink/Sink This holds a reference to the respective SubmodelElement used as data sink for live-data. This holds a reference to the respective SubmodelElement used as data sink for live-data.	[]	1

[Prop] SinkId	https://admin-shell.io/idta/AssetInterfacesMappingConfiguration/2/0/MappingConfiguration/Sink/SinkId This is a unique and non-empty identifier that facilitates data writing in the Lua transformation or establishes a relationship to a corresponding source when no transformation is given. It must only be unique with respect to the Sinks-list of the containing MappingConfiguration and not globally. This is a unique and non-empty identifier that facilitates data writing in the Lua transformation or establishes a relationship to a corresponding source when no transformation is given. It must only be unique with respect to the Sinks-list of the containing MappingConfiguration and not globally.	[String]	1

Annex A. Explanation of the used Table Formats

1. General

The tables used in this document try to outline information as concise 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 Submodel template 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 pieces of information in different rows of the same table cell. In this case, the first information is marked out by sharp brackets [] from the second piece of information. A special case are the semanticIds, which are marked out by the format: (type)(local)[idType]value.
- The types of SubmodelElements are abbreviated:

SME type	**SubmodelElement type**
Property	Property
MLP	MultiLanguageProperty
Range	Range
File	File
Blob	Blob
Ref	ReferenceElement
Rel	RelationshipElement
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: https://admin-shell.io/vdi/2770/1/0/DocumentId/Id. The attribute "type" (typically "ConceptDescription" and "(local)" or "GlobalReference") needs 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.
- 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. Lua and AIMC

1. Rationale behind Lua

Although alternative languages and notations are possible (e.g., Structured Text, Python, or expression-based DSLs), Lua is selected as the normative transformation language. Lua is a mature scripting language that has been around since 1993. It has been used in numerous software projects to provide the user the possibility to extend the behavior of the application. The possibility to sandbox Lua environments and mitigate exploitation risk of harmful code is another advantage. Furthermore, Lua is light-weight and there are wrappers for many available programming languages, such as C++, C#, Java and Python. Finally, Lua comes with a simple syntax with only few keywords and is therefore easy to comprehend. This makes Lua a practical choice for portable deployment of the DMP across edge, gateway, and server environments.

2. Prototypical Implementation of a Lua Processor

The following code snippet outlines how any external Lua script can be executed within Python. Even though the code does a lot to make the Lua environment secure, there is no guarantee that there are no exploits. The code is meant as educational content so only use it at your own risk!

```
from lupa import LuaRuntime

# Limit the memory Lua can utilize.
MAX_MEMORY_BYTES = 32 * 1024 * 1024  # 32 MB

# Limit CPU time Lua is allowed to use. Avoid runaway loops.
MAX_TIME_SECONDS = 2

# Remove unsafe python functions Lua could call.
lua = LuaRuntime(register_eval=False,
                    register_builtins=False,
                    unpack_returned_tuples=True,
                    max_memory=MAX_MEMORY_BYTES)

# Implement hook that is called once in a while to check if the time is used up.
lua.execute(f'''
    -- Hook that implements timeout
    local start = os.clock()
    debug.sethook(function ()
        if os.clock() - start > {MAX_TIME_SECONDS} then
            error("Lua-timeout after {MAX_TIME_SECONDS} second(s) CPU time!")
        end
    end, "", 1000)
''')

# Implement a whitelist of Lua modules that are allowed to be called.
lua.execute('''
    -- Whitelist: Safe modules
    local safe = {
        assert   = assert,
        error    = error,
```

```
        ipairs   = ipairs,
        next     = next,
        pairs    = pairs,
        pcall    = pcall,
        select   = select,
        tonumber = tonumber,
        tostring = tostring,
        type     = type,
        math     = math,
        table    = table,
    }

    -- Remove unsafe modules, except for os.
    local G = _G or _Env
    for k in pairs(G) do
        if safe[k] == nil then
            if k ~= "os" then
                G[k] = nil
            end
        end
    end

    -- Only allow os.clock in os module. Needed for hook.
    G.os = { clock = os.clock }
''')

lua_code = '''
function aimc_main(sources)
    local value_1 = sources["source1"]
    local value_2 = sources["source2"]

    return {
        ["Test1"] = value_1 + value_2,
        ["Test2"] = math.sqrt(value_1*value_1 + value_2*value_2)
    }
end
'''

lua.execute(lua_code)
sources = {'source1': 7, 'source2': 35}
lua_table = lua.table_from(sources)

result = dict(lua.globals().aimc_main(lua_table))
print(result)
```

Annex C. Changes to the Submodel Template

Changes between Version 1.0 and 2.0

- Added description to SMT AIMC
- Changed semantic-ID of SMT AIMC
- Added description to SML MappingConfigurations
- Added description to SMC MappingConfiguration
- Changed semantic-ID of SMC MappingConfiguration due to the new features
- Removed ReferenceElement InterfaceReference from SMC MappingConfiguration since the related AID Submodel can be inferred from each Source ReferenceElement
- Added Property DefaultPollingInterval to SMC MappingConfiguration
- Added Blob Transformation to SMC MappingConfiguration
- Removed SML MappingSourceSinkRelations from SMC MappingConfiguration
- Added SML Sources to SMC MappingConfiguration
- Added SML Sinks to SMC MappingConfiguration

Bibliography

[1] “Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group”, acatech, April 2013. [Online]. Available https://en.acatech.de/publication/recommendations-for-implementing-the-strategic-initiative-industrie-4-0-final-report-of-the-industrie-4-0-working-group/

[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., January 2016. [Online]. Available: https://www.bitkom.org/Bitkom/Publikationen/Implementation-Strategy-Industrie-40-Report-on-the-results-of-the-Industrie-40-Platform.html

[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/presse-medien/publikationen/beispiele-zur-verwaltungsschale-der-industrie-40-komponente-basisteil/

[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] “Specification of the Asset Administration Shell Part 1: Metamodel – IDTA Number: 01001-3-0-2”, March 2025, [Online]. Available: https://industrialdigitaltwin.org/wp-content/uploads/2025/03/IDTA-01001-3-0-2_SpecificationAssetAdministrationShell_Part1_Metamodel.pdf

[7] S. Kaebisch, M. McCool and E. Korkan, “Web of Things (WoT) Thing Description 1.1“, World Wide Web Consortium, 2023. [Online]. Available: https://www.w3.org/TR/wot-thing-description11/

[8] Modbus Organisation, “MODBUS Messaging on TCP/IP Implementation Guide V1.0b“, Modbus Organisation, 2006. [Online]. Available: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[9] OASIS, “MQTT Version 3.1.1 Plus Errata 01“, OASIS, December 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[10] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach and T. Berners-Lee, „RFC 2616 - Hypertext Transfer Protocol — HTTP/1.1“, 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616

[11] “Specification of the Asset Administration Shell Part 2: Application Programming Interfaces – IDTA Number: 01002-3-0-4”, April 2025, [Online]. Available: https://industrialdigitaltwin.org/wp-content/uploads/2025/04/IDTA-01002-3-0-4_SpecificationAssetAdministrationShell_Part2_API.pdf

[12] OPC Foundation, OPC Unified Architecture. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/

[13] ANSI/ASHRAE Standard 135-2020: BACnet – A Data Communication Protocol for Building Automation Networks. ASHRAE. 2020. Available: https://www.ashrae.org/technical-resources/bookstore/bacnet

[14] IO-LINK Interface and System, Version 1.1.3; IO-Link Community; June 2019.

[15] JSON Integration for IO-Link; IO-Link Community; March 2022.

[16] IO-Link Integration Edition 2: Specification for PROFINET. PROFIBUS Nutzerorganisation e. V.; February 2020.

[17] Asset Administration Shell Submodel Template; Asset Interfaces Description; Version 1.1, 2025, Industrial Digital Twin Association, Available: https://github.com/admin-shell-io/submodel-templates/tree/main/published/Asset%20Interfaces%20Description

[18] The Lua scripting language. [Online]. Available: https://www.lua.org/about.html

[19] Asset Administration Shell Submodel Template; Process Variables for Manufacturing Key Performance Indicators; Version 1.0, 2026, Industrial Digital Twin Association, Available: ToDo