aas-core3.0 Documentation

Release 1.0.4

Marko Ristin

Apr 16, 2024

CONTENTS:

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Introduction

This is a software development kit (SDK) to:

- manipulate,
- verify, and
- de/serialize to and from JSON and XML

… Asset Administration Shells based on the version 3.0V of the meta-model.

For a brief introduction, see *Getting Started*.

For a detailed documentation of the API, see *API*.

We document some of the design decisions in: *Design Decisions*.

1.2 Getting Started

Here's a quick intro to get you started with the SDK. See how you can:

- *Install the SDK*,
- *Programmatically create, get and set properties of an AAS model*,
- *Iterate over and transform a model*,
- *Verify a model*,
- *De/serialize a model from and to JSON*, and
- *De/serialize a model from and to XML*.

1.2.1 Installation

Activate your virtual environment.

Install the SDK by calling:

```
pip3 install aas-core3.0
```

1.2.2 Create, Get and Set Properties of an AAS Model

The module *aas_core3.types* contains all the data types of the meta-model. This includes enumerations, abstract and concrete classes.

The module *aas_core3.types* also contains visitors and transformers, but we will write more about them in *Iterate and Transform* section.

Creation

We use constructors to create an AAS model.

Usually you start bottom-up, all the way up to the *aas_core3.types.Environment*.

Getting and Setting Properties

All properties of the classes are modeled as Python properties.

After initialization of a class, you can directly get and modify its properties.

Getters with a Default Value

For optional properties which come with a default value, we provide special getters, `{property name}_or_default`. If the property is `None`, this getter will give you the default value. Otherwise, if the property is set, the actual value of the property will be returned.

For example, see *aas_core3.types.HasKind.kind_or_default()*.

Example: Create an Environment with a Submodel

Here is a very rudimentary example where we show how to create an environment which contains a submodel.

The submodel will contain two elements, a property and a blob.

```
import aas_core3.types as aas_types

# Create the first element
some_element = aas_types.Property(
    id_short="some_property",
    value_type=aas_types.DataTypeDefXSD.INT,
    value="1984"
)

# Create the second element
another_element = aas_types.Blob(
    id_short="some_blob",
    content_type="application/octet-stream",
    value=b'\xDE\xAD\xBE\xEF'
)

# You can directly access the element properties.
another_element.value = b'\xDE\xAD\xC0\xDE'
```

(continues on next page)

(continued from previous page)

```
# Nest the elements in a submodel
submodel = aas_types.Submodel(
    id="some-unique-global-identifier",
    submodel_elements=[
        some_element,
        another_element
    ]
)

# Now create the environment to wrap it all up
environment = aas_types.Environment(
    submodels=[submodel]
)

# You can access the properties from the children as well.
environment.submodels[0].submodel_elements[1].value = b'\xC0\x01\xCA\xFE'

# Now you can do something with the environment...
```

1.2.3 Iterate and Transform

The SDK provides various ways how you can loop through the elements of the model, and how these elements can be transformed. Each following section will look into one of the approaches.

over_X_or_empty

For all the optional lists, there is a corresponding `over_{property name}_or_empty` getter. It gives you an `Iterator`. If the property is not set, this getter will yield empty. Otherwise, it will yield from the actual property value.

For example, see *aas_core3.types.Environment.over_submodels_or_empty()*.

descend_once and descend

If you are writing a simple script and do not care about the performance, the SDK provides two methods in the most general interface *aas_core3.types.Class*, *descend_once()* and *descend()*, which you can use to loop through the instances.

Both *descend_once()* and *descend()* iterate over referenced children of an instance of *Class*. *descend_once()*, as it names suggests, stops after all the children has been iterated over. *descend()* continues recursively to grand-children, grand-grand-children *etc*.

Here is a short example how you can get all the properties from an environment whose ID-short starts with `another`:

```
import aas_core3.types as aas_types

# Prepare the environment
environment = aas_types.Environment(
    submodels=[
        aas_types.Submodel(
            id="some-unique-global-identifier",
```

(continues on next page)

(continued from previous page)

```
            submodel_elements=[
                aas_types.Property(
                    id_short="some_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1984"
                ),
                aas_types.Property(
                    id_short="another_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1985"
                ),
                aas_types.Property(
                    id_short="yet_another_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1986"
                )
            ]
        )
    ]
)

for something in environment.descend():
    if (
        isinstance(something, aas_types.Property)
        and "another" in something.id_short
    ):
        print(something.id_short)
```

```
another_property
yet_another_property
```

Iteration with *descend_once()* and *descend()* works well if the performance is irrelevant. However, if the performance matters, this is not a good approach. First, all the children will be visited (even though you need only a small subset). Second, you need to switch with `isinstance()` on the runtime type, which grows linearly in computational cost with the number of types you switch on.

Let's see in the next section how we could use a more efficient, but also a more complex approach.

Visitor

Visitor pattern is a common design pattern in software engineering. We will not explain the details of the pattern here as you can read about in the ample literature in books or in Internet.

The cornerstone of the visitor pattern is double dispatch: instead of casting to the desired type during the iteration, the method *aas_core3.types.Class.accept()* directly dispatches to the appropriate visitation method.

This allows us to spare runtime type switches and directly dispatch the execution. The SDK already implements *accept()* methods, so you only have to implement the visitor.

The visitor class has a visiting method for each class of the meta-model. In the SDK, we provide different flavors of the visitor abstract classes which you can readily implement:

- *AbstractVisitor* which needs all the visit methods to be implemented,
- *PassThroughVisitor* which visits all the elements and does nothing, and

- *AbstractVisitorWithContext* which propagates a context object along the iteration.

Let us re-write the above example related to *descend()* method with a visitor pattern:

```
import aas_core3.types as aas_types

class Visitor(aas_types.PassThroughVisitor):
    def visit_property(self, that: aas_types.Property):
        if "another" in that.id_short:
            print(that.id_short)

# Prepare the environment
environment = aas_types.Environment(
    submodels=[
        aas_types.Submodel(
            id="some-unique-global-identifier",
            submodel_elements=[
                aas_types.Property(
                    id_short="some_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1984"
                ),
                aas_types.Property(
                    id_short="another_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1985"
                ),
                aas_types.Property(
                    id_short="yet_another_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1986"
                )
            ]
        )
    ]
)

# Iterate
visitor = Visitor()
visitor.visit(environment)
```

Expected output:

```
another_property
yet_another_property
```

There are important differences to iteration with *descend()*:

- Due to double dispatch, we spare a cast. This is usually more efficient.
- The iteration logic in *descend()* lives very close to where it is executed. In contrast, the visitor needs to be defined as a separate class. While sometimes faster, writing the visitor makes the code less readable.

Descend or Visitor?

In general, people familiar with the visitor pattern and object-oriented programming will prefer, obviously, visitor class. People who like functional programming, generator expressions and ilks will prefer *descend()*.

It is difficult to discuss different tastes, so you should probably come up with explicit code guidelines in your code and stick to them.

Make sure you always profile before you sacrifice readability and blindly apply one or the other approach for performance reasons.

Transformer

A transformer pattern is an analogous to visitor pattern, where we "transform" the visited element into some other form (be it a string or a different object). It is very common in compiler design, where the abstract syntax tree is transformed into a different representation.

The SDK provides different flavors of a transformer:

- *AbstractTransformer*, where the model element is directly transformed into something, and
- *AbstractTransformerWithContext*, which propagates the context object along the transformations.

Usually you implement for each concrete class how it should be transformed. If you want to specify only a subset of transformations, and provide the default value for the remainder, the SDK provides *TransformerWithDefault* and *TransformerWithDefaultAndContext*.

We deliberately omit an example due to the length of the code. Please let us know by creating an issue if you would like to have an example here.

1.2.4 Verify

Our SDK allows you to verify that a model satisfies the constraints of the meta-model.

The verification logic is concentrated in the module *aas_core3.verification*, and all it takes is a call to *aas_core3.verification.verify()* function. The function *aas_core3.verification.verify()* will check that constraints in the given model element are satisfied, including the recursion into children elements. The function returns an iterator of *aas_core3.verification.Error*'s, which you can use for further processing (*e.g.*, report to the user).

Here is a short example snippet:

```
import aas_core3.types as aas_types
import aas_core3.verification as aas_verification

# Prepare the environment
environment = aas_types.Environment(
    submodels=[
        aas_types.Submodel(
            id="some-unique-global-identifier",
            submodel_elements=[
                aas_types.Property(
                    # The ID-shorts must be proper variable names,
                    # but there is a dash ("-") in this ID-short.
                    id_short = "some-Property",
                    value_type=aas_types.DataTypeDefXSD.INT,
```

(continues on next page)

(continued from previous page)

```
                    value="1984"
                )
            ]
        )
    ]
)

for error in aas_verification.verify(environment):
    print(f"{error.path}: {error.cause}")
```

Expected output:

```
.submodels[0].submodel_elements[0].id_short: ID-short of Referables shall only feature␣
↪letters, digits, underscore (``_``); starting mandatory with a letter. *I.e.* ``[a-zA-
↪Z][a-zA-Z0-9_]*``.
```

Limit the Number of Reported Errors

Since the function *aas_core3.verification.verify()* gives you an iterator, you can use `itertools` on it.

Here is a snippet which reports only the first 10 errors:

```
# ... code from above ...

import itertools

for error in itertools.islice(
        aas_verification.verify(environment),
        10
):
    print(f"{error.path}: {error.cause}")
```

Omitted Constraints

Not all constraints specified in the meta-model can be verified. Some constraints require external dependencies such as an AAS registry. Verifying the constraints with external dependencies is out-of-scope of our SDK, as we still lack standardized interfaces to those dependencies.

However, all the constraints which need no external dependency are verified. For a full list of exception, please see the description of the module *aas_core3.types*.

1.2.5 JSON De/serialization

Our SDK handles the de/serialization of the AAS models from and to JSON format through the module *aas_core3.jsonization*.

Serialize

To serialize, you call the function *aas_core3.jsonization.to_jsonable()* on an instance of *aas_core3.types.Environment* which will convert it to a JSON-able mapping.

Here is a snippet that converts the environment first into a JSON-able mapping, and next converts the JSON-able mapping to text:

```
import json

import aas_core3.types as aas_types
import aas_core3.jsonization as aas_jsonization

# Prepare the environment
environment = aas_types.Environment(
    submodels=[
        aas_types.Submodel(
            id="some-unique-global-identifier",
            submodel_elements=[
                aas_types.Property(
                    id_short = "some_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1984"
                )
            ]
        )
    ]
)

# Serialize to a JSON-able mapping
jsonable = aas_jsonization.to_jsonable(environment)

# Print the mapping as text
print(json.dumps(jsonable, indent=2))
```

Expected output:

```
{
  "submodels": [
    {
      "id": "some-unique-global-identifier",
      "submodelElements": [
        {
          "idShort": "some_property",
          "valueType": "xs:int",
          "value": "1984",
          "modelType": "Property"
        }
      ],
      "modelType": "Submodel"
    }
  ]
}
```

De-serialize

Our SDK can convert a JSON-able mapping back to an instance of *aas_core3.types.Environment*. To that end, you call the function *aas_core3.jsonization.environment_from_jsonable()*.

Here is an example snippet:

```
import json

import aas_core3.jsonization as aas_jsonization

text = """\
    {
      "submodels": [
        {
          "id": "some-unique-global-identifier",
          "submodelElements": [
            {
              "idShort": "someProperty",
              "valueType": "xs:boolean",
              "modelType": "Property"
            }
          ],
          "modelType": "Submodel"
        }
      ]
    }"""

jsonable = json.loads(text)

environment = aas_jsonization.environment_from_jsonable(
    jsonable
)

for something in environment.descend():
    print(type(something))
```

Expected output:

```
<class 'aas_core3.types.Submodel'>
<class 'aas_core3.types.Property'>
```

Errors

If there are any errors during the de-serialization, an *aas_core3.jsonization.DeserializationException* will be thrown. Errors occur whenever we encounter invalid JSON values. For example, this is the case when the de-serialization function expects a JSON object, but encounters a JSON array instead.

1.2.6 XML De/serialization

The code that de/serializes AAS models from and to XML documents lives in the module *aas_core3.xmlization*.

Serialize

You serialize the AAS model to XML-encoded text by calling the function *aas_core3.xmlization.to_str()*.

If you want the same text to be written incrementally to a `typing.TextIO` stream, you can use the function *aas_core3.xmlization.write()*.

Here is an example snippet:

```
import aas_core3.types as aas_types
import aas_core3.xmlization as aas_xmlization

# Prepare the environment
environment = aas_types.Environment(
    submodels=[
        aas_types.Submodel(
            id="some-unique-global-identifier",
            submodel_elements=[
                aas_types.Property(
                    id_short = "some_property",
                    value_type=aas_types.DataTypeDefXSD.INT,
                    value="1984"
                )
            ]
        )
    ]
)

# Serialize to an XML-encoded string
text = aas_xmlization.to_str(environment)

print(text)
```

Expected output:

```
<environment xmlns="https://admin-shell.io/aas/3/0"><submodels><submodel><id>some-unique-
↪global-identifier</id><submodelElements><property><idShort>some_property</idShort>
↪<valueType>xs:int</valueType><value>1984</value></property></submodelElements></
↪submodel></submodels></environment>
```

De-serialize

You can de-serialize an environment from XML coming from four different sources by using different functions:

- *aas_core3.xmlization.environment_from_iterparse()*, where a stream coming from `xml.etree.ElementTree.iterparse()` is expected with `events` set to `["start", "end"]`,
- *aas_core3.xmlization.environment_from_stream()*, which expects a textual stream behaving according to `typing.TextIO`,
- *aas_core3.xmlization.environment_from_file()*, which expects a path to the file containing the XML of the environment, or
- *aas_core3.xmlization.environment_from_str()*, which de-serialized the environment from an XML-encoded string.

Here is a snippet which parses XML as text and then de-serializes it into an instance of *Environment*:

```
import aas_core3.xmlization as aas_xmlization

text = (
    "<environment xmlns=\"https://admin-shell.io/aas/3/0\">" +
    "<submodels><submodel>" +
    "<id>some-unique-global-identifier</id>" +
    "<submodelElements><property><idShort>someProperty</idShort>" +
    "<valueType>xs:boolean</valueType></property></submodelElements>" +
    "</submodel></submodels></environment>"
)

environment = aas_xmlization.environment_from_str(text)

for something in environment.descend():
    print(type(something))
```

Expected output:

```
<class 'aas_core3.types.Submodel'>
<class 'aas_core3.types.Property'>
```

Errors

If the XML document comes in an unexpected form, our SDK throws a *aas_core3.xmlization.DeserializationException*. This can happen, for example, if unexpected XML elements or XML attributes are encountered, or an expected XML element is missing.

1.3 API

This is the documentation automatically generated from the source code.

1.3.1 aas_core3.common

Provide common functions shared among the modules.

aas_core3.common.**assert_never**(*value: NoReturn*) → NoReturn

Signal to mypy to perform an exhaustive matching.

Please see the following page for more details: https://hakibenita.com/python-mypy-exhaustive-checking

1.3.2 aas_core3.constants

Provide constant values of the meta-model.

aas_core3.constants.**VALID_CATEGORIES_FOR_DATA_ELEMENT: Set[str] = {'CONSTANT', 'PARAMETER', 'VARIABLE'}**

Categories for *types.DataElement* as defined in *Constraint AASd-090*

aas_core3.constants.**GENERIC_FRAGMENT_KEYS: Set[*KeyTypes*] = {KeyTypes.FRAGMENT_REFERENCE}**

Enumeration of all identifiable elements within an asset administration shell.

aas_core3.constants.**GENERIC_GLOBALLY_IDENTIFIABLES: Set[*KeyTypes*] = {KeyTypes.GLOBAL_REFERENCE}**

Enumeration of different key value types within a key.

aas_core3.constants.**AAS_IDENTIFIABLES: Set[*KeyTypes*] = {<KeyTypes.SUBMODEL: 'Submodel'>, <KeyTypes.IDENTIFIABLE: 'Identifiable'>, <KeyTypes.CONCEPT_DESCRIPTION: 'ConceptDescription'>, <KeyTypes.ASSET_ADMINISTRATION_SHELL: 'AssetAdministrationShell'>}**

Enumeration of different key value types within a key.

aas_core3.constants.**AAS_SUBMODEL_ELEMENTS_AS_KEYS: Set[*KeyTypes*] = {<KeyTypes.OPERATION: 'Operation'>, <KeyTypes.RANGE: 'Range'>, <KeyTypes.ENTITY: 'Entity'>, <KeyTypes.FILE: 'File'>, <KeyTypes.ANNOTATED_RELATIONSHIP_ELEMENT: 'AnnotatedRelationshipElement'>, <KeyTypes.BLOB: 'Blob'>, <KeyTypes.PROPERTY: 'Property'>, <KeyTypes.MULTI_LANGUAGE_PROPERTY: 'MultiLanguageProperty'>, <KeyTypes.CAPABILITY: 'Capability'>, <KeyTypes.RELATIONSHIP_ELEMENT: 'RelationshipElement'>, <KeyTypes.SUBMODEL_ELEMENT: 'SubmodelElement'>, <KeyTypes.SUBMODEL_ELEMENT_LIST: 'SubmodelElementList'>, <KeyTypes.EVENT_ELEMENT: 'EventElement'>, <KeyTypes.REFERENCE_ELEMENT: 'ReferenceElement'>, <KeyTypes.SUBMODEL_ELEMENT_COLLECTION: 'SubmodelElementCollection'>, <KeyTypes.BASIC_EVENT_ELEMENT: 'BasicEventElement'>, <KeyTypes.DATA_ELEMENT: 'DataElement'>}**

Enumeration of all submodel elements within an asset administration shell.

aas_core3.constants.**AAS_REFERABLE_NON_IDENTIFIABLES: Set[*KeyTypes*] = {<KeyTypes.OPERATION: 'Operation'>, <KeyTypes.RANGE: 'Range'>, <KeyTypes.ENTITY: 'Entity'>, <KeyTypes.FILE: 'File'>, <KeyTypes.ANNOTATED_RELATIONSHIP_ELEMENT: 'AnnotatedRelationshipElement'>, <KeyTypes.BLOB: 'Blob'>, <KeyTypes.PROPERTY: 'Property'>, <KeyTypes.MULTI_LANGUAGE_PROPERTY: 'MultiLanguageProperty'>, <KeyTypes.CAPABILITY: 'Capability'>, <KeyTypes.RELATIONSHIP_ELEMENT: 'RelationshipElement'>, <KeyTypes.SUBMODEL_ELEMENT: 'SubmodelElement'>, <KeyTypes.SUBMODEL_ELEMENT_LIST: 'SubmodelElementList'>, <KeyTypes.EVENT_ELEMENT: 'EventElement'>, <KeyTypes.REFERENCE_ELEMENT: 'ReferenceElement'>, <KeyTypes.SUBMODEL_ELEMENT_COLLECTION: 'SubmodelElementCollection'>, <KeyTypes.BASIC_EVENT_ELEMENT: 'BasicEventElement'>, <KeyTypes.DATA_ELEMENT: 'DataElement'>}**

Enumeration of different fragment key value types within a key.

aas_core3.constants.**AAS_REFERABLES: Set[*KeyTypes*] = {<KeyTypes.FILE: 'File'>, <KeyTypes.ANNOTATED_RELATIONSHIP_ELEMENT: 'AnnotatedRelationshipElement'>, <KeyTypes.PROPERTY: 'Property'>, <KeyTypes.IDENTIFIABLE: 'Identifiable'>, <KeyTypes.OPERATION: 'Operation'>, <KeyTypes.BLOB: 'Blob'>, <KeyTypes.RELATIONSHIP_ELEMENT: 'RelationshipElement'>, <KeyTypes.CAPABILITY: 'Capability'>, <KeyTypes.REFERENCE_ELEMENT: 'ReferenceElement'>, <KeyTypes.SUBMODEL_ELEMENT_COLLECTION: 'SubmodelElementCollection'>, <KeyTypes.EVENT_ELEMENT: 'EventElement'>, <KeyTypes.CONCEPT_DESCRIPTION: 'ConceptDescription'>, <KeyTypes.MULTI_LANGUAGE_PROPERTY: 'MultiLanguageProperty'>, <KeyTypes.SUBMODEL_ELEMENT: 'SubmodelElement'>, <KeyTypes.SUBMODEL_ELEMENT_LIST: 'SubmodelElementList'>, <KeyTypes.REFERABLE: 'Referable'>, <KeyTypes.DATA_ELEMENT: 'DataElement'>, <KeyTypes.RANGE: 'Range'>, <KeyTypes.ASSET_ADMINISTRATION_SHELL: 'AssetAdministrationShell'>, <KeyTypes.ENTITY: 'Entity'>, <KeyTypes.SUBMODEL: 'Submodel'>, <KeyTypes.BASIC_EVENT_ELEMENT: 'BasicEventElement'>}**

Enumeration of referables. We need this to check that model references refer to a Referable. For example, the observed attribute of the Basic Event Element object must be a model reference to a Referable.

aas_core3.constants.**GLOBALLY_IDENTIFIABLES: Set[*KeyTypes*] = {<KeyTypes.CONCEPT_DESCRIPTION: 'ConceptDescription'>, <KeyTypes.IDENTIFIABLE: 'Identifiable'>, <KeyTypes.SUBMODEL: 'Submodel'>, <KeyTypes.GLOBAL_REFERENCE: 'GlobalReference'>, <KeyTypes.ASSET_ADMINISTRATION_SHELL: 'AssetAdministrationShell'>}**

Enumeration of all referable elements within an asset administration shell

aas_core3.constants.**FRAGMENT_KEYS: Set[*KeyTypes*] = {<KeyTypes.OPERATION: 'Operation'>, <KeyTypes.RANGE: 'Range'>, <KeyTypes.ENTITY: 'Entity'>, <KeyTypes.FILE: 'File'>, <KeyTypes.ANNOTATED_RELATIONSHIP_ELEMENT: 'AnnotatedRelationshipElement'>, <KeyTypes.FRAGMENT_REFERENCE: 'FragmentReference'>, <KeyTypes.BLOB: 'Blob'>, <KeyTypes.PROPERTY: 'Property'>, <KeyTypes.MULTI_LANGUAGE_PROPERTY: 'MultiLanguageProperty'>, <KeyTypes.CAPABILITY: 'Capability'>, <KeyTypes.RELATIONSHIP_ELEMENT: 'RelationshipElement'>, <KeyTypes.SUBMODEL_ELEMENT: 'SubmodelElement'>, <KeyTypes.SUBMODEL_ELEMENT_LIST: 'SubmodelElementList'>, <KeyTypes.EVENT_ELEMENT: 'EventElement'>, <KeyTypes.REFERENCE_ELEMENT: 'ReferenceElement'>, <KeyTypes.SUBMODEL_ELEMENT_COLLECTION: 'SubmodelElementCollection'>, <KeyTypes.BASIC_EVENT_ELEMENT: 'BasicEventElement'>, <KeyTypes.DATA_ELEMENT: 'DataElement'>}**

Enumeration of different key value types within a key.

aas_core3.constants.**DATA_TYPE_IEC_61360_FOR_PROPERTY_OR_VALUE: Set[*DataTypeIEC61360*] = {<DataTypeIEC61360.TIMESTAMP: 'TIMESTAMP'>, <DataTypeIEC61360.STRING: 'STRING'>, <DataTypeIEC61360.RATIONAL_MEASURE: 'RATIONAL_MEASURE'>, <DataTypeIEC61360.INTEGER_COUNT: 'INTEGER_COUNT'>, <DataTypeIEC61360.RATIONAL: 'RATIONAL'>, <DataTypeIEC61360.TIME: 'TIME'>, <DataTypeIEC61360.REAL_CURRENCY: 'REAL_CURRENCY'>, <DataTypeIEC61360.INTEGER_MEASURE: 'INTEGER_MEASURE'>, <DataTypeIEC61360.REAL_MEASURE: 'REAL_MEASURE'>, <DataTypeIEC61360.STRING_TRANSLATABLE: 'STRING_TRANSLATABLE'>, <DataTypeIEC61360.INTEGER_CURRENCY: 'INTEGER_CURRENCY'>, <DataTypeIEC61360.BOOLEAN: 'BOOLEAN'>, <DataTypeIEC61360.REAL_COUNT: 'REAL_COUNT'>, <DataTypeIEC61360.DATE: 'DATE'>}**

IEC 61360 data types for concept descriptions categorized with PROPERTY or VALUE.

aas_core3.constants.**DATA_TYPE_IEC_61360_FOR_REFERENCE: Set[*DataTypeIEC61360*] = {<DataTypeIEC61360.IRDI: 'IRDI'>, <DataTypeIEC61360.IRI: 'IRI'>, <DataTypeIEC61360.STRING: 'STRING'>}**

IEC 61360 data types for concept descriptions categorized with REFERENCE.

aas_core3.constants.**DATA_TYPE_IEC_61360_FOR_DOCUMENT: Set[*DataTypeIEC61360*] = {<DataTypeIEC61360.BLOB: 'BLOB'>, <DataTypeIEC61360.HTML: 'HTML'>, <DataTypeIEC61360.FILE: 'FILE'>}**

IEC 61360 data types for concept descriptions categorized with DOCUMENT.

aas_core3.constants.**IEC_61360_DATA_TYPES_WITH_UNIT: Set[*DataTypeIEC61360*] = {<DataTypeIEC61360.RATIONAL_MEASURE: 'RATIONAL_MEASURE'>, <DataTypeIEC61360.REAL_CURRENCY: 'REAL_CURRENCY'>, <DataTypeIEC61360.INTEGER_MEASURE: 'INTEGER_MEASURE'>, <DataTypeIEC61360.REAL_MEASURE: 'REAL_MEASURE'>, <DataTypeIEC61360.INTEGER_CURRENCY: 'INTEGER_CURRENCY'>}**

These data types imply that the unit is defined in the data specification.

1.3.3 aas_core3.jsonization

Provide de/serialization of AAS classes to/from JSON.

We can not use one-pass deserialization for JSON since the object properties do not have fixed order, and hence we can not read `modelType` property ahead of the remaining properties.

class aas_core3.jsonization.**PropertySegment**(*instance: Mapping[str, Any]*, *name: str*)

Represent a property on a path to the erroneous value.

__init__(*instance: Mapping[str, Any]*, *name: str*) → None

Initialize with the given values.

instance: Final[Mapping[str, Any]]

Instance that contains the property

name: Final[str]

Name of the property

class aas_core3.jsonization.**IndexSegment**(*container: Iterable[Any]*, *index: int*)

Represent an index access on a path to the erroneous value.

__init__(*container: Iterable[Any]*, *index: int*) → None

Initialize with the given values.

container: Final[Iterable[Any]]

Container that contains the item

index: Final[int]

Index of the item

class aas_core3.jsonization.**Path**

Represent the relative path to the erroneous value.

__init__() → None

Initialize as an empty path.

property segments: Sequence[Union[*PropertySegment*, *IndexSegment*]]

Get the segments of the path.

__str__() → str

Return str(self).

exception aas_core3.jsonization.**DeserializationException**(*cause: str*)

Signal that the JSON de-serialization could not be performed.

__init__(*cause: str*) → None

Initialize with the given `cause` and an empty path.

cause: Final[str]

Human-readable explanation of the exception's cause

path: Final[*Path*]

Relative path to the erroneous value

aas_core3.jsonization.**has_semantics_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *HasSemantics*

Parse an instance of *types.HasSemantics* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Concrete instance of *types.HasSemantics*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**extension_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Extension*

Parse an instance of *types.Extension* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Extension*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**has_extensions_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *HasExtensions*

Parse an instance of *types.HasExtensions* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Concrete instance of *types.HasExtensions*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**referable_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Referable*

Parse an instance of *types.Referable* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Concrete instance of *types.Referable*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`identifiable_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Identifiable*

Parse an instance of *types.Identifiable* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.Identifiable*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`modelling_kind_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ModellingKind*

Convert the JSON-able structure `jsonable` to a literal of *types.ModellingKind*.

Parameters
jsonable – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`has_kind_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *HasKind*

Parse an instance of *types.HasKind* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.HasKind*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`has_data_specification_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *HasDataSpecification*

Parse an instance of *types.HasDataSpecification* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.HasDataSpecification*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`administrative_information_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AdministrativeInformation*

Parse an instance of *types.AdministrativeInformation* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.AdministrativeInformation*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`qualifiable_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Qualifiable*

Parse an instance of *types.Qualifiable* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Concrete instance of *types.Qualifiable*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`qualifier_kind_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *QualifierKind*

Convert the JSON-able structure `jsonable` to a literal of *types.QualifierKind*.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`qualifier_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Qualifier*

Parse an instance of *types.Qualifier* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Qualifier*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`asset_administration_shell_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AssetAdministrationShell*

Parse an instance of *types.AssetAdministrationShell* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.AssetAdministrationShell*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`asset_information_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AssetInformation*

Parse an instance of *types.AssetInformation* from the JSON-able structure `jsonable`.

Parameters
`jsonable` – structure to be parsed

Returns
Parsed instance of *types.AssetInformation*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`resource_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Resource*

Parse an instance of *types.Resource* from the JSON-able structure `jsonable`.

Parameters
`jsonable` – structure to be parsed

Returns
Parsed instance of *types.Resource*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`asset_kind_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AssetKind*

Convert the JSON-able structure `jsonable` to a literal of *types.AssetKind*.

Parameters
`jsonable` – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`specific_asset_id_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *SpecificAssetID*

Parse an instance of *types.SpecificAssetID* from the JSON-able structure `jsonable`.

Parameters
`jsonable` – structure to be parsed

Returns
Parsed instance of *types.SpecificAssetID*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**submodel_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Submodel*

Parse an instance of *types.Submodel* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.Submodel*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**submodel_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *SubmodelElement*

Parse an instance of *types.SubmodelElement* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.SubmodelElement*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**relationship_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *RelationshipElement*

Parse an instance of *types.RelationshipElement* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.RelationshipElement*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**aas_submodel_elements_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AASSubmodelElements*

Convert the JSON-able structure `jsonable` to a literal of *types.AASSubmodelElements*.

Parameters
jsonable – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**submodel_element_list_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *SubmodelElementList*

Parse an instance of *types.SubmodelElementList* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.SubmodelElementList*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**submodel_element_collection_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *SubmodelElementCollection*

Parse an instance of *types.SubmodelElementCollection* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.SubmodelElementCollection*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**data_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *DataElement*

Parse an instance of *types.DataElement* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Concrete instance of *types.DataElement*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**property_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Property*

Parse an instance of *types.Property* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Property*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**multi_language_property_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *MultiLanguageProperty*

Parse an instance of *types.MultiLanguageProperty* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.MultiLanguageProperty*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**range_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Range*

Parse an instance of types.Range from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.Range

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**reference_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ReferenceElement*

Parse an instance of types.ReferenceElement from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.ReferenceElement

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**blob_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Blob*

Parse an instance of types.Blob from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.Blob

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**file_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *File*

Parse an instance of types.File from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.File

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**annotated_relationship_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AnnotatedRelationshipElement*

Parse an instance of types.AnnotatedRelationshipElement from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.AnnotatedRelationshipElement*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**entity_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Entity*

Parse an instance of *types.Entity* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Entity*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**entity_type_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *EntityType*

Convert the JSON-able structure jsonable to a literal of *types.EntityType*.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**direction_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Direction*

Convert the JSON-able structure jsonable to a literal of *types.Direction*.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**state_of_event_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *StateOfEvent*

Convert the JSON-able structure jsonable to a literal of *types.StateOfEvent*.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**event_payload_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *EventPayload*

Parse an instance of *types.EventPayload* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.EventPayload*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**event_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *EventElement*

Parse an instance of *types.EventElement* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.EventElement*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**basic_event_element_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *BasicEventElement*

Parse an instance of *types.BasicEventElement* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.BasicEventElement*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**operation_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Operation*

Parse an instance of *types.Operation* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.Operation*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**operation_variable_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *OperationVariable*

Parse an instance of *types.OperationVariable* from the JSON-able structure `jsonable`.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.OperationVariable*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**capability_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Capability*

Parse an instance of *types.Capability* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Capability*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**concept_description_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ConceptDescription*

Parse an instance of *types.ConceptDescription* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.ConceptDescription*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**reference_types_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ReferenceTypes*

Convert the JSON-able structure jsonable to a literal of *types.ReferenceTypes*.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**reference_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Reference*

Parse an instance of *types.Reference* from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of *types.Reference*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**key_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Key*

Parse an instance of *types.Key* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.Key*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**key_types_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *KeyTypes*

Convert the JSON-able structure `jsonable` to a literal of *types.KeyTypes*.

Parameters
jsonable – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**data_type_def_xsd_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *DataTypeDefXSD*

Convert the JSON-able structure `jsonable` to a literal of *types.DataTypeDefXSD*.

Parameters
jsonable – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**abstract_lang_string_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *AbstractLangString*

Parse an instance of *types.AbstractLangString* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Concrete instance of *types.AbstractLangString*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**lang_string_name_type_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LangStringNameType*

Parse an instance of *types.LangStringNameType* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.LangStringNameType*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`lang_string_text_type_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LangStringTextType*

Parse an instance of *types.LangStringTextType* from the JSON-able structure `jsonable`.

Parameters
`jsonable`¶ – structure to be parsed

Returns
Parsed instance of *types.LangStringTextType*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`environment_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *Environment*

Parse an instance of *types.Environment* from the JSON-able structure `jsonable`.

Parameters
`jsonable`¶ – structure to be parsed

Returns
Parsed instance of *types.Environment*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`data_specification_content_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *DataSpecificationContent*

Parse an instance of *types.DataSpecificationContent* from the JSON-able structure `jsonable`.

Parameters
`jsonable`¶ – structure to be parsed

Returns
Concrete instance of *types.DataSpecificationContent*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`embedded_data_specification_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *EmbeddedDataSpecification*

Parse an instance of *types.EmbeddedDataSpecification* from the JSON-able structure `jsonable`.

Parameters
`jsonable`¶ – structure to be parsed

Returns
Parsed instance of *types.EmbeddedDataSpecification*

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**data_type_iec_61360_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *DataTypeIEC61360*

Convert the JSON-able structure jsonable to a literal of types.DataTypeIEC61360.

Parameters
jsonable¶ – JSON-able structure to be parsed

Returns
parsed literal

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**level_type_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LevelType*

Parse an instance of types.LevelType from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.LevelType

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**value_reference_pair_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ValueReferencePair*

Parse an instance of types.ValueReferencePair from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.ValueReferencePair

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**value_list_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *ValueList*

Parse an instance of types.ValueList from the JSON-able structure jsonable.

Parameters
jsonable¶ – structure to be parsed

Returns
Parsed instance of types.ValueList

Raise
DeserializationException if unexpected jsonable

aas_core3.jsonization.**lang_string_preferred_name_type_iec_61360_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LangStringPreferredNameTypeIEC61360*

Parse an instance of *types.LangStringPreferredNameTypeIEC61360* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.LangStringPreferredNameTypeIEC61360*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**lang_string_short_name_type_iec_61360_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LangStringShortNameTypeIEC61360*

Parse an instance of *types.LangStringShortNameTypeIEC61360* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.LangStringShortNameTypeIEC61360*

Raise
DeserializationException if unexpected `jsonable`

aas_core3.jsonization.**lang_string_definition_type_iec_61360_from_jsonable**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *LangStringDefinitionTypeIEC61360*

Parse an instance of *types.LangStringDefinitionTypeIEC61360* from the JSON-able structure `jsonable`.

Parameters
jsonable – structure to be parsed

Returns
Parsed instance of *types.LangStringDefinitionTypeIEC61360*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`data_specification_iec_61360_from_jsonable`**(*jsonable: Union[bool, int, float, str, Sequence[Any], Mapping[str, Any]]*) → *DataSpecificationIEC61360*

Parse an instance of *types.DataSpecificationIEC61360* from the JSON-able structure `jsonable`.

Parameters
`jsonable` – structure to be parsed

Returns
Parsed instance of *types.DataSpecificationIEC61360*

Raise
DeserializationException if unexpected `jsonable`

`aas_core3.jsonization.`**`to_jsonable`**(*that: Class*) → Union[bool, int, float, str, List[Any], MutableMapping[str, Any]]

Convert `that` to a JSON-able structure.

Parameters
`that` – AAS data to be recursively converted to a JSON-able structure

Returns
JSON-able structure which can be further encoded with, *e.g.*, `json`

1.3.4 aas_core3.stringification

De-serialize enumerations from string representations.

`aas_core3.stringification.`**`modelling_kind_from_str`**(*text: str*) → Optional[*ModellingKind*]

Parse `text` as string representation of `aas_core3.ModellingKind`.

If `text` is not a valid string representation of a literal of `aas_core3.ModellingKind`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.ModellingKind` or `None`, if *text* invalid.

`aas_core3.stringification.`**`qualifier_kind_from_str`**(*text: str*) → Optional[*QualifierKind*]

Parse `text` as string representation of `aas_core3.QualifierKind`.

If `text` is not a valid string representation of a literal of `aas_core3.QualifierKind`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.QualifierKind` or `None`, if *text* invalid.

`aas_core3.stringification.`**`asset_kind_from_str`**(*text: str*) → Optional[*AssetKind*]

Parse `text` as string representation of `aas_core3.AssetKind`.

If `text` is not a valid string representation of a literal of `aas_core3.AssetKind`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.AssetKind` or `None`, if `text` invalid.

`aas_core3.stringification.`**`aas_submodel_elements_from_str`**(*text: str*) → Optional[*AASSubmodelElements*]

Parse `text` as string representation of `aas_core3.AASSubmodelElements`.

If `text` is not a valid string representation of a literal of `aas_core3.AASSubmodelElements`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.AASSubmodelElements` or `None`, if `text` invalid.

`aas_core3.stringification.`**`entity_type_from_str`**(*text: str*) → Optional[*EntityType*]

Parse `text` as string representation of `aas_core3.EntityType`.

If `text` is not a valid string representation of a literal of `aas_core3.EntityType`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.EntityType` or `None`, if `text` invalid.

`aas_core3.stringification.`**`direction_from_str`**(*text: str*) → Optional[*Direction*]

Parse `text` as string representation of `aas_core3.Direction`.

If `text` is not a valid string representation of a literal of `aas_core3.Direction`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.Direction` or `None`, if `text` invalid.

`aas_core3.stringification.`**`state_of_event_from_str`**(*text: str*) → Optional[*StateOfEvent*]

Parse `text` as string representation of `aas_core3.StateOfEvent`.

If `text` is not a valid string representation of a literal of `aas_core3.StateOfEvent`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.StateOfEvent` or `None`, if `text` invalid.

`aas_core3.stringification.`**`reference_types_from_str`**(*text: str*) → Optional[*ReferenceTypes*]

Parse `text` as string representation of `aas_core3.ReferenceTypes`.

If `text` is not a valid string representation of a literal of `aas_core3.ReferenceTypes`, return `None`.

Parameters
`text` – to be parsed

Returns
the corresponding literal of `aas_core3.ReferenceTypes` or `None`, if `text` invalid.

`aas_core3.stringification.`**`key_types_from_str`**(*text: str*) → Optional[*KeyTypes*]

Parse `text` as string representation of `aas_core3.KeyTypes`.

If `text` is not a valid string representation of a literal of `aas_core3.KeyTypes`, return `None`.

Parameters
text¶ – to be parsed

Returns
the corresponding literal of `aas_core3.KeyTypes` or `None`, if `text` invalid.

`aas_core3.stringification.`**`data_type_def_xsd_from_str`**`(`*text: str*`)` → Optional[*DataTypeDefXSD*]

Parse `text` as string representation of `aas_core3.DataTypeDefXSD`.

If `text` is not a valid string representation of a literal of `aas_core3.DataTypeDefXSD`, return `None`.

Parameters
text¶ – to be parsed

Returns
the corresponding literal of `aas_core3.DataTypeDefXSD` or `None`, if `text` invalid.

`aas_core3.stringification.`**`data_type_iec_61360_from_str`**`(`*text: str*`)` → Optional[*DataTypeIEC61360*]

Parse `text` as string representation of `aas_core3.DataTypeIEC61360`.

If `text` is not a valid string representation of a literal of `aas_core3.DataTypeIEC61360`, return `None`.

Parameters
text¶ – to be parsed

Returns
the corresponding literal of `aas_core3.DataTypeIEC61360` or `None`, if `text` invalid.

1.3.5 aas_core3.types

Provide an implementation of the Asset Administration Shell (AAS) V3.0.

The presented version of the Metamodel is related to the work of aas-core-works, which can be found here: https://github.com/aas-core-works.

The presented content is neither related to the IDTA nor Plattform Industrie 4.0 and does not represent an official publication.

We diverge from the book in the following points.

We did not implement the following constraints as they are too general and can not be formalized as part of the core library, but affects external components such as AAS registry or AAS server:

- *Constraint AASd-022*

We did not implement the following constraints since they depend on registry and de-referencing of `Reference` objects:

- *Constraint AASd-006*
- *Constraint AASd-007*
- *Constraint AASc-3a-003*

Some constraints are not enforceable as they depend on the wider context such as language understanding, so we could not formalize them:

- *Constraint AASd-012*: This constraint requires that the texts inside `Multi_language_property` shall have the same meanings in the separate languages. This cannot be tested.
- *Constraint AASd-116*: In the book, *Constraint AASd-116* imposes a case-insensitive equality against `globalAssetId`. This is culturally-dependent, and depends on the system settings. For example, the case-folding for the letters “i” and “I” is different in Turkish from English.

  We implement the constraint as case-sensitive instead to allow for interoperability across different culture settings.

Furthermore, we diverge from the book in the following points regarding the enumerations. We have to implement subsets of enumerations as sets as common programming languages do not support inheritance of enumerations. The relationship between the properties and the sets is defined through invariants. This causes the following divergences:

- We decided therefore to remove the enumeration `DataTypeDefRDF` and keep only *DataTypeDefXSD* as enumeration. Otherwise, we would have to write redundant invariants all over the meta-model because `DataTypeDefRDF` is actually never used in any type definition.
- The enumeration *AASSubmodelElements* is used in two different contexts. One context is the definition of key types in a reference. Another context is the definition of element types in a *SubmodelElementList*.

  To avoid confusion, we introduce two separate enumerations for the separate contexts. Firstly, a set of *KeyTypes*, `constants.AAS_SUBMODEL_ELEMENTS_AS_KEYS` to represent the first context (key type in a reference). Secondly, the enumeration *AASSubmodelElements* is kept as designator for *SubmodelElementList.type_value_list_element*.
- The specification introduces several types of `Lang_string_set`. These types differ between the allowed length of their text inside the singular `Lang_string` objects. Since the native representation of `Lang_string_set` as `List` of `Lang_string` is required by specification, it is impossible to introduce separate `Lang_string_set` types. Therefore, the distinction is drawn here between the `Lang_string` types.

  `DefinitionTypeIEC61360` is represented as a `List` of *LangStringDefinitionTypeIEC61360*

  `MultiLanguageNameType` is represented as a `List` of *LangStringNameType*

  `PreferredNameTypeIEC61360` is represented as a `List` of *LangStringPreferredNameTypeIEC61360*

  `ShortNameTypeIEC61360` is represented as a `List` of *LangStringShortNameTypeIEC61360*

  `MultiLanguageTextType` is represented as a `List` of *LangStringTextType*

  Furthermore, since `Lang_string` is not used anywhere, we rename it to *AbstractLangString*.

Concerning the data specifications, we embed them within *HasDataSpecification* instead of referencing them *via* an external reference. The working group decided to change the rules for serialization *after* the book was published. The data specifications are critical in applications, but there is no possibility to access them through a data channel as they are not part of an environment.

class aas_core3.types.**Class**

Represent the most general class of an AAS model.

abstract descend_once() → Iterator[*Class*]

Iterate over all the instances referenced from this one.

abstract descend() → Iterator[*Class*]

Iterate recursively over all the instances referenced from this one.

abstract accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

Parameters
visitor¶ – to be dispatched

abstract accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance with `context`.

Parameters

- **visitor**¶ – to be dispatched
- **context**¶ – of the visitation

abstract transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

Parameters
transformer¶ – to be dispatched

Returns
transformed self

abstract transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance with `context`.

Parameters
transformer¶ – to be dispatched

Returns
transformed self

class aas_core3.types.**HasSemantics**(*semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*)

Element that can have a semantic definition plus some supplemental semantic definitions.

Constraint AASd-118
If there are ID *supplemental_semantic_ids* defined then there shall be also a main semantic ID *semantic_id*.

over_supplemental_semantic_ids_or_empty() → Iterator[*Reference*]

Yield from *supplemental_semantic_ids* if set.

__init__(*semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*) → None

Initialize with the given values.

semantic_id: Optional[*Reference*]

Identifier of the semantic definition of the element. It is called semantic ID of the element or also main semantic ID of the element.

Note: It is recommended to use a global reference.

---

supplemental_semantic_ids: Optional[List[*Reference*]]

Identifier of a supplemental semantic definition of the element. It is called supplemental semantic ID of the element.

Note: It is recommended to use a global reference.

---

class aas_core3.types.**Extension**(*name: str*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *value_type: Optional[*DataTypeDefXSD*] = None*, *value: Optional[str] = None*, *refers_to: Optional[List[*Reference*]] = None*)

Single extension of an element.

over_refers_to_or_empty() → Iterator[*Reference*]

Yield from *refers_to* if set.

value_type_or_default() → *DataTypeDefXSD*

Return the *value_type* if set, or the default otherwise.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*name: str*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *value_type: Optional[*DataTypeDefXSD*] = None*, *value: Optional[str] = None*, *refers_to: Optional[List[*Reference*]] = None*) → None

Initialize with the given values.

name: str

Name of the extension.

Constraint AASd-077
The name of an extension (Extension/name) within *HasExtensions* needs to be unique.

value_type: Optional[*DataTypeDefXSD*]

Type of the value of the extension.

Default: *DataTypeDefXSD.STRING*

value: Optional[str]

Value of the extension

refers_to: Optional[List[*Reference*]]

Reference to an element the extension refers to.

class aas_core3.types.**HasExtensions**(*extensions: Optional[List[*Extension*]] = None*)

Element that can be extended by proprietary extensions.

Note: Extensions are proprietary, i.e. they do not support global interoperability.

over_extensions_or_empty() → Iterator[*Extension*]

Yield from *extensions* if set.

__init__(*extensions: Optional[List[*Extension*]] = None*) → None

Initialize with the given values.

extensions: Optional[List[*Extension*]]

An extension of the element.

class aas_core3.types.**Referable**(*extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*)

An element that is referable by its *id_short*.

This ID is not globally unique. This ID is unique within the name space of the element.

Constraint AASd-022
id_short of non-identifiable referables within the same name space shall be unique (case-sensitive).

over_display_name_or_empty() → Iterator[*LangStringNameType*]

Yield from *display_name* if set.

over_description_or_empty() → Iterator[*LangStringTextType*]

Yield from *description* if set.

__init__(*extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*) → None

Initialize with the given values.

id_short: Optional[str]

In case of identifiables this attribute is a short name of the element. In case of referable this ID is an identifying string of the element within its name space.

Note: In case the element is a property and the property has a semantic definition (*HasSemantics.semantic_id*) conformant to IEC61360 the *id_short* is typically identical to the short name in English.

---

display_name: Optional[List[*LangStringNameType*]]

Display name. Can be provided in several languages.

category: Optional[str]

The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.

Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.

---

description: Optional[List[*LangStringTextType*]]

Description or comments on the element.

The description can be provided in several languages.

If no description is defined, then the definition of the concept description that defines the semantics of the element is used.

Additional information can be provided, e.g., if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates are provided.

class aas_core3.types.**Identifiable**(*id: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *administration: Optional[AdministrativeInformation] = None*)

An element that has a globally unique identifier.

__init__(*id: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *administration: Optional[AdministrativeInformation] = None*) → None

Initialize with the given values.

id: str

The globally unique identification of the element.

administration: Optional[*AdministrativeInformation*]

Administrative information of an identifiable element.

Note: Some of the administrative information like the version number might need to be part of the identification.

---

class aas_core3.types.**ModellingKind**(*value*)

Enumeration for denoting whether an element is a template or an instance.

TEMPLATE = 'Template'

Specification of the common features of a structured element in sufficient detail that such a instance can be instantiated using it

INSTANCE = 'Instance'

Concrete, clearly identifiable element instance. Its creation and validation may be guided by a corresponding element template.

class aas_core3.types.**HasKind**(*kind: Optional[ModellingKind] = None*)

An element with a kind is an element that can either represent a template or an instance.

Default for an element is that it is representing an instance.

kind_or_default() → *ModellingKind*

Return *kind* if set, and the default otherwise.

__init__(*kind: Optional[ModellingKind] = None*) → None

Initialize with the given values.

kind: Optional[*ModellingKind*]

Kind of the element: either type or instance.

Default: *ModellingKind.INSTANCE*

class aas_core3.types.**HasDataSpecification**(*embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*)

Element that can be extended by using data specification templates.

A data specification template defines a named set of additional attributes an element may or shall have. The data specifications used are explicitly specified with their global ID.

over_embedded_data_specifications_or_empty() → Iterator[*EmbeddedDataSpecification*]

Yield from `embedded_data_specifications` if set.

__init__(*embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*) → None

Initialize with the given values.

embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]]

Embedded data specification.

class aas_core3.types.**AdministrativeInformation**(*embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *version: Optional[str] = None*, *revision: Optional[str] = None*, *creator: Optional[Reference] = None*, *template_id: Optional[str] = None*)

Administrative meta-information for an element like version information.

Constraint AASd-005
If `version` is not specified then also `revision` shall be unspecified. This means, a revision requires a version. If there is no version there is no revision neither. Revision is optional.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *version: Optional[str] = None*, *revision: Optional[str] = None*, *creator: Optional[Reference] = None*, *template_id: Optional[str] = None*) → None

Initialize with the given values.

version: Optional[str]

Version of the element.

revision: Optional[str]

Revision of the element.

creator: Optional[*Reference*]

The subject ID of the subject responsible for making the element.

template_id: Optional[str]

Identifier of the template that guided the creation of the element.

Note: In case of a submodel the *template_id* is the identifier of the submodel template ID that guided the creation of the submodel

---

Note: The *template_id* is not relevant for validation in Submodels. For validation the `Submodel.semantic_id` shall be used.

Note: Usage of *template_id* is not restricted to submodel instances. So also the creation of submodel templates can be guided by another submodel template.

---

class aas_core3.types.**Qualifiable**(*qualifiers: Optional[List[*Qualifier*]] = None*)

The value of a qualifiable element may be further qualified by one or more qualifiers.

Constraint AASd-119

If any *Qualifier.kind* value of *qualifiers* is equal to *QualifierKind.TEMPLATE_QUALIFIER* and the qualified element inherits from *HasKind* then the qualified element shall be of kind Template (*HasKind.kind* = *ModellingKind.TEMPLATE*).

Note: This constraint is checked at *Submodel*.

---

over_qualifiers_or_empty() → Iterator[*Qualifier*]

Yield from *qualifiers* if set.

__init__(*qualifiers: Optional[List[*Qualifier*]] = None*) → None

Initialize with the given values.

qualifiers: Optional[List[*Qualifier*]]

Additional qualification of a qualifiable element.

Constraint AASd-021

Every qualifiable can only have one qualifier with the same *Qualifier.type*.

class aas_core3.types.**QualifierKind**(*value*)

Enumeration for kinds of qualifiers.

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

VALUE_QUALIFIER = 'ValueQualifier'

qualifies the value of the element and can change during run-time.

Value qualifiers are only applicable to elements with kind *ModellingKind.INSTANCE*.

CONCEPT_QUALIFIER = 'ConceptQualifier'

qualifies the semantic definition the element is referring to (*HasSemantics.semantic_id*)

TEMPLATE_QUALIFIER = 'TemplateQualifier'

qualifies the elements within a specific submodel on concept level.

Template qualifiers are only applicable to elements with kind *ModellingKind.TEMPLATE*.

class aas_core3.types.**Qualifier**(*type: str*, *value_type:* DataTypeDefXSD, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *kind: Optional[*QualifierKind*] = None*, *value: Optional[str] = None*, *value_id: Optional[*Reference*] = None*)

A qualifier is a type-value-pair that makes additional statements w.r.t. the value of the element.

Constraint AASd-006
If both the *value* and the *value_id* of a *Qualifier* are present then the *value* needs to be identical to the value of the referenced coded value in *value_id*.

Constraint AASd-020
The value of *value* shall be consistent to the data type as defined in *value_type*.

kind_or_default() → *QualifierKind*

Return *kind* if set, and the default otherwise.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*type: str*, *value_type:* DataTypeDefXSD, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *kind: Optional[*QualifierKind*] = None*, *value: Optional[str] = None*, *value_id: Optional[*Reference*] = None*) → None

Initialize with the given values.

type: str

The qualifier *type* describes the type of the qualifier that is applied to the element.

value_type: *DataTypeDefXSD*

Data type of the qualifier value.

kind: Optional[*QualifierKind*]

The qualifier kind describes the kind of the qualifier that is applied to the element.

Default: *QualifierKind.CONCEPT_QUALIFIER*

value: Optional[str]

The qualifier value is the value of the qualifier.

value_id: Optional[*Reference*]

Reference to the global unique ID of a coded value.

Note: It is recommended to use a global reference.

---

class aas_core3.types.**AssetAdministrationShell**(*id: str*, *asset_information:* AssetInformation, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *administration: Optional[*AdministrativeInformation*] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *derived_from: Optional[*Reference*] = None*, *submodels: Optional[List[*Reference*]] = None*)

An asset administration shell.

over_submodels_or_empty() → Iterator[*Reference*]

Yield from *submodels* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*id: str*, *asset_information:* AssetInformation, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *administration: Optional[*AdministrativeInformation*] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *derived_from: Optional[*Reference*] = None*, *submodels: Optional[List[*Reference*]] = None*) → None

Initialize with the given values.

derived_from: Optional[*Reference*]

The reference to the AAS the AAS was derived from.

asset_information: *AssetInformation*

Meta-information about the asset the AAS is representing.

submodels: Optional[List[*Reference*]]

References to submodels of the AAS.

A submodel is a description of an aspect of the asset the AAS is representing.

The asset of an AAS is typically described by one or more submodels.

Temporarily no submodel might be assigned to the AAS.

class aas_core3.types.**AssetInformation**(*asset_kind:* AssetKind, *global_asset_id: Optional[str] = None*, *specific_asset_ids: Optional[List[*SpecificAssetID*]] = None*, *asset_type: Optional[str] = None*, *default_thumbnail: Optional[*Resource*] = None*)

In *AssetInformation* identifying meta data of the asset that is represented by an AAS is defined.

The asset may either represent an asset type or an asset instance.

The asset has a globally unique identifier plus – if needed – additional domain specific (proprietary) identifiers. However, to support the corner case of very first phase of lifecycle where a stabilised/constant_set global asset identifier does not already exist, the corresponding attribute *global_asset_id* is optional.

Constraint AASd-116
`globalAssetId` is a reserved key. If used as value for *SpecificAssetID.name* then *SpecificAssetID.value* shall be identical to *global_asset_id*.

Note: *Constraint AASd-116* is important to enable a generic search across global and specific asset IDs.

Note: In the book, *Constraint AASd-116* imposes a case-insensitive equality against `globalAssetId`. This is culturally-dependent, and depends on the system settings. For example, the case-folding for the letters "i" and "I" is different in Turkish from English.

We implement the constraint as case-sensitive instead to allow for interoperability across different culture settings.

Constraint AASd-131
For *AssetInformation* either the *global_asset_id* shall be defined or at least one item in *specific_asset_ids*.

over_specific_asset_ids_or_empty() → Iterator[*SpecificAssetID*]

Yield from *specific_asset_ids* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*asset_kind:* AssetKind, *global_asset_id: Optional[str] = None*, *specific_asset_ids: Optional[List[*SpecificAssetID*]] = None*, *asset_type: Optional[str] = None*, *default_thumbnail: Optional[*Resource*] = None*) → None

Initialize with the given values.

asset_kind: *AssetKind*

Denotes whether the Asset is of kind *AssetKind.TYPE* or *AssetKind.INSTANCE*.

global_asset_id: `Optional[str]`

Global identifier of the asset the AAS is representing.

This attribute is required as soon as the AAS is exchanged via partners in the life cycle of the asset. In a first phase of the life cycle the asset might not yet have a global ID but already an internal identifier. The internal identifier would be modelled via *specific_asset_ids*.

Note: This is a global reference.

---

specific_asset_ids: `Optional[List[`*SpecificAssetID*`]]`

Additional domain-specific, typically proprietary identifier for the asset like e.g., serial number etc.

asset_type: `Optional[str]`

In case *asset_kind* is applicable the *asset_type* is the asset ID of the type asset of the asset under consideration as identified by *global_asset_id*.

Note: In case *asset_kind* is "Instance" than the *asset_type* denotes which "Type" the asset is of. But it is also possible to have an *asset_type* of an asset of kind "Type".

default_thumbnail: Optional[*Resource*]

Thumbnail of the asset represented by the Asset Administration Shell.

Used as default.

class aas_core3.types.**Resource**(*path: str*, *content_type: Optional[str] = None*)

Resource represents an address to a file (a locator). The value is an URI that can represent an absolute or relative path

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*path: str*, *content_type: Optional[str] = None*) → None

Initialize with the given values.

path: str

Path and name of the resource (with file extension).

The path can be absolute or relative.

content_type: Optional[str]

Content type of the content of the file.

The content type states which file extensions the file can have.

class aas_core3.types.**AssetKind**(*value*)

Enumeration for denoting whether an asset is a type asset or an instance asset.

TYPE = 'Type'

Type asset

INSTANCE = 'Instance'

Instance asset

NOT_APPLICABLE = 'NotApplicable'

Neither a type asset nor an instance asset

class aas_core3.types.**SpecificAssetID**(*name: str*, *value: str*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *external_subject_id: Optional[Reference] = None*)

A specific asset ID describes a generic supplementary identifying attribute of the asset.

The specific asset ID is not necessarily globally unique.

Constraint AASd-133
external_subject_id shall be an external reference, i.e. *Reference.type = ReferenceTypes.EXTERNAL_REFERENCE*.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*name: str*, *value: str*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *external_subject_id: Optional[Reference] = None*) → None

Initialize with the given values.

name: str

Name of the identifier

value: str

The value of the specific asset identifier with the corresponding name.

external_subject_id: Optional[*Reference*]

The (external) subject the key belongs to or has meaning to.

Note: This is a global reference.

---

class aas_core3.types.**Submodel**(*id: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *administration: Optional[AdministrativeInformation] = None*, *kind: Optional[ModellingKind] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *submodel_elements: Optional[List[SubmodelElement]] = None*)

A submodel defines a specific aspect of the asset represented by the AAS.

A submodel is used to structure the digital representation and technical functionality of an Administration Shell into distinguishable parts. Each submodel refers to a well-defined domain or subject matter. Submodels can become standardized and, thus, become submodels templates.

over_submodel_elements_or_empty() → Iterator[*SubmodelElement*]

Yield from `submodel_elements` if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*id: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *administration: Optional[AdministrativeInformation] = None*, *kind: Optional[ModellingKind] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *submodel_elements: Optional[List[SubmodelElement]] = None*) → None

Initialize with the given values.

submodel_elements: Optional[List[*SubmodelElement*]]

A submodel consists of zero or more submodel elements.

class aas_core3.types.**SubmodelElement**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*)

A submodel element is an element suitable for the description and differentiation of assets.

It is recommended to add a *HasSemantics.semantic_id* to a submodel element.

Constraint AASd-129
If any *Qualifier.kind* value of `qualifiers` (attribute qualifier inherited via Qualifiable) is equal to *QualifierKind.TEMPLATE_QUALIFIER* then the submodel element shall be part of a submodel template, i.e. a Submodel with `Submodel.kind` (attribute kind inherited via *HasKind*) value is equal to *ModellingKind.TEMPLATE*.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*) → None

Initialize with the given values.

category: Optional[str]

The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.

Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.

---

id_short: Optional[str]

In case of identifiables this attribute is a short name of the element. In case of referable this ID is an identifying string of the element within its name space.

Note: In case the element is a property and the property has a semantic definition (*HasSemantics.semantic_id*) conformant to IEC61360 the *id_short* is typically identical to the short name in English.

---

display_name: Optional[List[*LangStringNameType*]]

Display name. Can be provided in several languages.

description: Optional[List[*LangStringTextType*]]

Description or comments on the element.

The description can be provided in several languages.

If no description is defined, then the definition of the concept description that defines the semantics of the element is used.

Additional information can be provided, e.g., if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates are provided.

extensions: Optional[List[*Extension*]]

An extension of the element.

class aas_core3.types.**RelationshipElement**(*first:* Reference, *second:* Reference, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*)

A relationship element is used to define a relationship between two elements being either referable (model reference) or external (global reference).

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*first:* Reference, *second:* Reference, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*) → None

Initialize with the given values.

first: *Reference*

Reference to the first element in the relationship taking the role of the subject.

second: *Reference*

Reference to the second element in the relationship taking the role of the object.

class aas_core3.types.**AASSubmodelElements**(*value*)

Enumeration of all possible elements of a *SubmodelElementList*.

ANNOTATED_RELATIONSHIP_ELEMENT = 'AnnotatedRelationshipElement'

BASIC_EVENT_ELEMENT = 'BasicEventElement'

BLOB = 'Blob'

CAPABILITY = 'Capability'

DATA_ELEMENT = 'DataElement'

ENTITY = 'Entity'

EVENT_ELEMENT = 'EventElement'

FILE = 'File'

MULTI_LANGUAGE_PROPERTY = 'MultiLanguageProperty'

OPERATION = 'Operation'

PROPERTY = 'Property'

RANGE = 'Range'

REFERENCE_ELEMENT = 'ReferenceElement'

RELATIONSHIP_ELEMENT = 'RelationshipElement'

SUBMODEL_ELEMENT = 'SubmodelElement'

SUBMODEL_ELEMENT_LIST = 'SubmodelElementList'

SUBMODEL_ELEMENT_COLLECTION = 'SubmodelElementCollection'

class aas_core3.types.**SubmodelElementList**(*type_value_list_element: AASSubmodelElements*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *order_relevant: Optional[bool] = None*, *semantic_id_list_element: Optional[Reference] = None*, *value_type_list_element: Optional[DataTypeDefXSD] = None*, *value: Optional[List[SubmodelElement]] = None*)

A submodel element list is an ordered list of submodel elements.

The numbering starts with zero (0).

Constraint AASd-107
If a first level child element in a *SubmodelElementList* has a *HasSemantics.semantic_id* it shall be identical to *semantic_id_list_element*.

Constraint AASd-114
If two first level child elements in a *SubmodelElementList* have a *HasSemantics.semantic_id* then they shall be identical.

Constraint AASd-115
If a first level child element in a *SubmodelElementList* does not specify a *HasSemantics.semantic_id* then the value is assumed to be identical to *semantic_id_list_element*.

Constraint AASd-120
The `id_short` of a *SubmodelElement* being a direct child of a *SubmodelElementList* shall not be specified.

Constraint AASd-108
All first level child elements in a *SubmodelElementList* shall have the same submodel element type as specified in *type_value_list_element*.

Constraint AASd-109
If *type_value_list_element* is equal to *AASSubmodelElements.PROPERTY* or *AASSubmodelElements.RANGE* *value_type_list_element* shall be set and all first level child elements in the *SubmodelElementList* shall have the value type as specified in *value_type_list_element*.

over_value_or_empty() → Iterator[*SubmodelElement*]

Yield from *value* if set.

order_relevant_or_default() → bool

Return *order_relevant* if set, and the default otherwise.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*type_value_list_element: AASSubmodelElements*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *order_relevant: Optional[bool] = None*, *semantic_id_list_element: Optional[Reference] = None*, *value_type_list_element: Optional[DataTypeDefXSD] = None*, *value: Optional[List[SubmodelElement]] = None*) → None

Initialize with the given values.

type_value_list_element: *AASSubmodelElements*

The submodel element type of the submodel elements contained in the list.

order_relevant: **Optional[bool]**

Defines whether order in list is relevant. If *order_relevant* = `False` then the list is representing a set or a bag.

Default: `True`

semantic_id_list_element: **Optional[*Reference*]**

Semantic ID the submodel elements contained in the list match to.

Note: It is recommended to use a global reference.

---

value_type_list_element: **Optional[*DataTypeDefXSD*]**

The value type of the submodel element contained in the list.

value: **Optional[List[*SubmodelElement*]]**

Submodel element contained in the list.

The list is ordered.

class aas_core3.types.**SubmodelElementCollection**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[List[SubmodelElement]] = None*)

A submodel element collection is a kind of struct, i.e. a a logical encapsulation of multiple named values. It has a fixed number of submodel elements.

over_value_or_empty() → Iterator[*SubmodelElement*]

Yield from *value* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *value: Optional[List[*SubmodelElement*]] = None*) → None

Initialize with the given values.

value: Optional[List[*SubmodelElement*]]

Submodel element contained in the collection.

class aas_core3.types.**DataElement**(*extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*)

A data element is a submodel element that is not further composed out of other submodel elements.

A data element is a submodel element that has a value. The type of value differs for different subtypes of data elements.

Constraint AASd-090
For data elements *category* shall be one of the following values: `CONSTANT`, `PARAMETER` or `VARIABLE`.

Default: `VARIABLE`

category_or_default() → str

Return the *category* if set or the default value otherwise.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*) → None

Initialize with the given values.

category: Optional[str]

The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.

Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.

---

id_short: Optional[str]

In case of identifiables this attribute is a short name of the element. In case of referable this ID is an identifying string of the element within its name space.

Note: In case the element is a property and the property has a semantic definition (*HasSemantics.semantic_id*) conformant to IEC61360 the *id_short* is typically identical to the short name in English.

---

display_name: Optional[List[*LangStringNameType*]]

Display name. Can be provided in several languages.

description: Optional[List[*LangStringTextType*]]

Description or comments on the element.

The description can be provided in several languages.

If no description is defined, then the definition of the concept description that defines the semantics of the element is used.

Additional information can be provided, e.g., if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates are provided.

extensions: Optional[List[*Extension*]]

An extension of the element.

semantic_id: Optional[*Reference*]

Identifier of the semantic definition of the element. It is called semantic ID of the element or also main semantic ID of the element.

Note: It is recommended to use a global reference.

---

supplemental_semantic_ids: Optional[List[*Reference*]]

Identifier of a supplemental semantic definition of the element. It is called supplemental semantic ID of the element.

Note: It is recommended to use a global reference.

---

qualifiers: Optional[List[*Qualifier*]]

Additional qualification of a qualifiable element.

Constraint AASd-021
Every qualifiable can only have one qualifier with the same *Qualifier.type*.

embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]]

Embedded data specification.

class aas_core3.types.**Property**(*value_type: DataTypeDefXSD*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[str] = None*, *value_id: Optional[Reference] = None*)

A property is a data element that has a single value.

Constraint AASd-007
If both, the *value* and the *value_id* are present then the value of *value* needs to be identical to the value of the referenced coded value in *value_id*.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*value_type: DataTypeDefXSD*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[str] = None*, *value_id: Optional[Reference] = None*) → None

Initialize with the given values.

value_type: *DataTypeDefXSD*

Data type of the value

value: Optional[str]

The value of the property instance.

value_id: Optional[*Reference*]

Reference to the global unique ID of a coded value.

Note: It is recommended to use a global reference.

---

class aas_core3.types.**MultiLanguageProperty**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[List[LangStringTextType]] = None*, *value_id: Optional[Reference] = None*)

A property is a data element that has a multi-language value.

Constraint AASd-012
If both the *value* and the *value_id* are present then for each string in a specific language the meaning must be the same as specified in *value_id*.

over_value_or_empty() → Iterator[*LangStringTextType*]

Yield from *value* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[List[LangStringTextType]] = None*, *value_id: Optional[Reference] = None*) → None

Initialize with the given values.

value: Optional[List[*LangStringTextType*]]

The value of the property instance.

value_id: Optional[*Reference*]

Reference to the global unique ID of a coded value.

Note: It is recommended to use a global reference.

---

class aas_core3.types.**Range**(*value_type: DataTypeDefXSD*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *min: Optional[str] = None*, *max: Optional[str] = None*)

A range data element is a data element that defines a range with min and max.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*value_type:* DataTypeDefXSD, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *min: Optional[str] = None*, *max: Optional[str] = None*) → None

Initialize with the given values.

value_type: *DataTypeDefXSD*

Data type of the min und max

min: Optional[str]

The minimum value of the range.

If the min value is missing, then the value is assumed to be negative infinite.

max: Optional[str]

The maximum value of the range.

If the max value is missing, then the value is assumed to be positive infinite.

class aas_core3.types.**ReferenceElement**(*extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *value: Optional[*Reference*] = None*)

A reference element is a data element that defines a logical reference to another element within the same or another AAS or a reference to an external object or entity.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[Reference] = None*) → None

Initialize with the given values.

value: Optional[*Reference*]

Global reference to an external object or entity or a logical reference to another element within the same or another AAS (i.e. a model reference to a Referable).

class aas_core3.types.**Blob**(*content_type: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[bytes] = None*)

A *Blob* is a data element that represents a file that is contained with its source code in the value attribute.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*content_type: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[bytes] = None*) → None

Initialize with the given values.

content_type: str

Content type of the content of the *Blob*.

The content type (MIME type) states which file extensions the file can have.

Valid values are content types like e.g. `application/json`, `application/xls`, `image/jpg`.

The allowed values are defined as in RFC2046.

value: Optional[bytes]

The value of the *Blob* instance of a blob data element.

Note: In contrast to the file property the file content is stored directly as value in the *Blob* data element.

---

class aas_core3.types.**File**(*content_type: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[str] = None*)

A File is a data element that represents an address to a file (a locator).

The value is an URI that can represent an absolute or relative path.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*content_type: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *value: Optional[str] = None*) → None

Initialize with the given values.

content_type: str

Content type of the content of the file.

The content type states which file extensions the file can have.

value: Optional[str]

Path and name of the referenced file (with file extension).

The path can be absolute or relative.

class aas_core3.types.**AnnotatedRelationshipElement**(*first: Reference*, *second: Reference*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *annotations: Optional[List[DataElement]] = None*)

An annotated relationship element is a relationship element that can be annotated with additional data elements.

over_annotations_or_empty() → Iterator[*DataElement*]

Yield from *annotations* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*first:* Reference, *second:* Reference, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *annotations: Optional[List[*DataElement*]] = None*) → None

Initialize with the given values.

annotations: Optional[List[*DataElement*]]

A data element that represents an annotation that holds for the relationship between the two elements

class aas_core3.types.**Entity**(*entity_type:* EntityType, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *statements: Optional[List[*SubmodelElement*]] = None*, *global_asset_id: Optional[str] = None*, *specific_asset_ids: Optional[List[*SpecificAssetID*]] = None*)

An entity is a submodel element that is used to model entities.

Constraint AASd-014
Either the attribute *global_asset_id* or *specific_asset_ids* of an *Entity* must be set if *entity_type* is set to *EntityType.SELF_MANAGED_ENTITY*. They are not existing otherwise.

over_statements_or_empty() → Iterator[*SubmodelElement*]

Yield from *statements* if set.

over_specific_asset_ids_or_empty() → Iterator[*SpecificAssetID*]

Yield from *specific_asset_ids* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*entity_type:* EntityType, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *statements: Optional[List[*SubmodelElement*]] = None*, *global_asset_id: Optional[str] = None*, *specific_asset_ids: Optional[List[*SpecificAssetID*]] = None*) → None

Initialize with the given values.

statements: Optional[List[*SubmodelElement*]]

Describes statements applicable to the entity by a set of submodel elements, typically with a qualified value.

entity_type: *EntityType*

Describes whether the entity is a co-managed entity or a self-managed entity.

global_asset_id: Optional[str]

Global identifier of the asset the entity is representing.

Note: This is a global reference.

---

specific_asset_ids: Optional[List[*SpecificAssetID*]]

Reference to a specific asset ID representing a supplementary identifier of the asset represented by the Asset Administration Shell.

class aas_core3.types.**EntityType**(*value*)

Enumeration for denoting whether an entity is a self-managed entity or a co-managed entity.

CO_MANAGED_ENTITY = 'CoManagedEntity'

For co-managed entities there is no separate AAS. Co-managed entities need to be part of a self-managed entity.

SELF_MANAGED_ENTITY = 'SelfManagedEntity'

Self-Managed Entities have their own AAS but can be part of the bill of material of a composite self-managed entity.

The asset of an I4.0 Component is a self-managed entity per definition.

class aas_core3.types.**Direction**(*value*)

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

INPUT = 'input'

Input direction.

OUTPUT = 'output'

Output direction

class aas_core3.types.**StateOfEvent**(*value*)

State of an event

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

ON = 'on'

Event is on

OFF = 'off'

Event is off.

class aas_core3.types.**EventPayload**(*source: Reference*, *observable_reference: Reference*, *time_stamp: str*, *source_semantic_id: Optional[Reference] = None*, *observable_semantic_id: Optional[Reference] = None*, *topic: Optional[str] = None*, *subject_id: Optional[Reference] = None*, *payload: Optional[bytes] = None*)

Defines the necessary information of an event instance sent out or received.

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

---

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*source: Reference*, *observable_reference: Reference*, *time_stamp: str*, *source_semantic_id: Optional[Reference] = None*, *observable_semantic_id: Optional[Reference] = None*, *topic: Optional[str] = None*, *subject_id: Optional[Reference] = None*, *payload: Optional[bytes] = None*) → None

Initialize with the given values.

source: *Reference*

Reference to the source event element, including identification of *AssetAdministrationShell*, *Submodel*, *SubmodelElement*'s.

observable_reference: *Reference*

Reference to the referable, which defines the scope of the event.

Can be *AssetAdministrationShell*, *Submodel* or *SubmodelElement*.

time_stamp: str

Timestamp in UTC, when this event was triggered.

source_semantic_id: Optional[*Reference*]

HasSemantics.semantic_id of the source event element, if available

Note: It is recommended to use a global reference.

observable_semantic_id: Optional[*Reference*]

HasSemantics.semantic_id of the referable which defines the scope of the event, if available.

Note: It is recommended to use a global reference.

topic: Optional[str]

Information for the outer message infrastructure for scheduling the event to the respective communication channel.

subject_id: Optional[*Reference*]

Subject, who/which initiated the creation.

Note: This is an external reference.

payload: Optional[bytes]

Event specific payload.

class aas_core3.types.**EventElement**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*)

An event element.

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

`__init__`(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*) → None

Initialize with the given values.

category: Optional[str]

The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.

Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.

id_short: Optional[str]

In case of identifiables this attribute is a short name of the element. In case of referable this ID is an identifying string of the element within its name space.

Note: In case the element is a property and the property has a semantic definition (*HasSemantics.semantic_id*) conformant to IEC61360 the *id_short* is typically identical to the short name in English.

display_name: Optional[List[*LangStringNameType*]]

Display name. Can be provided in several languages.

description: Optional[List[*LangStringTextType*]]

Description or comments on the element.

The description can be provided in several languages.

If no description is defined, then the definition of the concept description that defines the semantics of the element is used.

Additional information can be provided, e.g., if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates are provided.

extensions: Optional[List[*Extension*]]

An extension of the element.

semantic_id: Optional[*Reference*]

Identifier of the semantic definition of the element. It is called semantic ID of the element or also main semantic ID of the element.

Note: It is recommended to use a global reference.

supplemental_semantic_ids: Optional[List[*Reference*]]

Identifier of a supplemental semantic definition of the element. It is called supplemental semantic ID of the element.

Note: It is recommended to use a global reference.

---

qualifiers: Optional[List[*Qualifier*]]

Additional qualification of a qualifiable element.

Constraint AASd-021
Every qualifiable can only have one qualifier with the same *Qualifier.type*.

embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]]

Embedded data specification.

class aas_core3.types.**BasicEventElement**(*observed: Reference*, *direction: Direction*, *state: StateOfEvent*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *message_topic: Optional[str] = None*, *message_broker: Optional[Reference] = None*, *last_update: Optional[str] = None*, *min_interval: Optional[str] = None*, *max_interval: Optional[str] = None*)

A basic event element.

Note: This element is experimental and therefore may be subject to change or may be removed completely in future versions of the meta-model.

---

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*observed:* Reference, *direction:* Direction, *state:* StateOfEvent, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *semantic_id: Optional[*Reference*] = None*, *supplemental_semantic_ids: Optional[List[*Reference*]] = None*, *qualifiers: Optional[List[*Qualifier*]] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *message_topic: Optional[str] = None*, *message_broker: Optional[*Reference*] = None*, *last_update: Optional[str] = None*, *min_interval: Optional[str] = None*, *max_interval: Optional[str] = None*) → None

Initialize with the given values.

observed: *Reference*

Reference to the *Referable*, which defines the scope of the event. Can be *AssetAdministrationShell*, *Submodel*, or *SubmodelElement*.

Reference to a referable, e.g., a data element or a submodel, that is being observed.

direction: *Direction*

Direction of event.

Can be `{ Input, Output }`.

state: *StateOfEvent*

State of event.

Can be `{ On, Off }`.

message_topic: Optional[str]

Information for the outer message infrastructure for scheduling the event to the respective communication channel.

message_broker: Optional[*Reference*]

Information, which outer message infrastructure shall handle messages for the *EventElement*. Refers to a *Submodel*, *SubmodelElementList*, *SubmodelElementCollection* or *Entity*, which contains *DataElement*'s describing the proprietary specification for the message broker.

Note: For different message infrastructure, e.g., OPC UA or MQTT or AMQP, this proprietary specification could be standardized by having respective Submodels.

---

last_update: Optional[str]

Timestamp in UTC, when the last event was received (input direction) or sent (output direction).

min_interval: Optional[str]

For input direction, reports on the maximum frequency, the software entity behind the respective Referable can handle input events.

For output events, specifies the maximum frequency of outputting this event to an outer infrastructure.

Might be not specified, that is, there is no minimum interval.

max_interval: Optional[str]

For input direction: not applicable.

For output direction: maximum interval in time, the respective Referable shall send an update of the status of the event, even if no other trigger condition for the event was not met.

Might be not specified, that is, there is no maximum interval

class aas_core3.types.**Operation**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *input_variables: Optional[List[OperationVariable]] = None*, *output_variables: Optional[List[OperationVariable]] = None*, *inoutput_variables: Optional[List[OperationVariable]] = None*)

An operation is a submodel element with input and output variables.

Constraint AASd-134
For an *Operation* the *Referable.id_short* of all *OperationVariable.value*'s in *input_variables*, *output_variables* and *inoutput_variables* shall be unique.

over_input_variables_or_empty() → Iterator[*OperationVariable*]

Yield from *input_variables* if set.

over_output_variables_or_empty() → Iterator[*OperationVariable*]

Yield from *output_variables* if set.

over_inoutput_variables_or_empty() → Iterator[*OperationVariable*]

Yield from *inoutput_variables* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *input_variables: Optional[List[OperationVariable]] = None*, *output_variables: Optional[List[OperationVariable]] = None*, *inoutput_variables: Optional[List[OperationVariable]] = None*) → None

Initialize with the given values.

input_variables: Optional[List[*OperationVariable*]]

Input parameter of the operation.

output_variables: Optional[List[*OperationVariable*]]

Output parameter of the operation.

inoutput_variables: Optional[List[*OperationVariable*]]

Parameter that is input and output of the operation.

class aas_core3.types.**OperationVariable**(*value: SubmodelElement*)

The value of an operation variable is a submodel element that is used as input and/or output variable of an operation.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor: AbstractVisitor*) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor: AbstractVisitorWithContext[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer: AbstractTransformer[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer: AbstractTransformerWithContext[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*value: SubmodelElement*) → None

Initialize with the given values.

value: *SubmodelElement*

Describes an argument or result of an operation via a submodel element

class aas_core3.types.**Capability**(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*)

A capability is the implementation-independent description of the potential of an asset to achieve a certain effect in the physical or virtual world.

Note: The *semantic_id* of a capability is typically an ontology. Thus, reasoning on capabilities is enabled.

---

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *semantic_id: Optional[Reference] = None*, *supplemental_semantic_ids: Optional[List[Reference]] = None*, *qualifiers: Optional[List[Qualifier]] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*) → None

Initialize with the given values.

category: `Optional[str]`

The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.

Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.

id_short: Optional[str]

In case of identifiables this attribute is a short name of the element. In case of referable this ID is an identifying string of the element within its name space.

Note: In case the element is a property and the property has a semantic definition (*HasSemantics.semantic_id*) conformant to IEC61360 the *id_short* is typically identical to the short name in English.

display_name: Optional[List[*LangStringNameType*]]

Display name. Can be provided in several languages.

description: Optional[List[*LangStringTextType*]]

Description or comments on the element.

The description can be provided in several languages.

If no description is defined, then the definition of the concept description that defines the semantics of the element is used.

Additional information can be provided, e.g., if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates are provided.

extensions: Optional[List[*Extension*]]

An extension of the element.

semantic_id: Optional[*Reference*]

Identifier of the semantic definition of the element. It is called semantic ID of the element or also main semantic ID of the element.

Note: It is recommended to use a global reference.

supplemental_semantic_ids: Optional[List[*Reference*]]

Identifier of a supplemental semantic definition of the element. It is called supplemental semantic ID of the element.

Note: It is recommended to use a global reference.

qualifiers: Optional[List[*Qualifier*]]

Additional qualification of a qualifiable element.

Constraint AASd-021
Every qualifiable can only have one qualifier with the same *Qualifier.type*.

embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]]

Embedded data specification.

class aas_core3.types.**ConceptDescription**(*id: str*, *extensions: Optional[List[Extension]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[LangStringNameType]] = None*, *description: Optional[List[LangStringTextType]] = None*, *administration: Optional[AdministrativeInformation] = None*, *embedded_data_specifications: Optional[List[EmbeddedDataSpecification]] = None*, *is_case_of: Optional[List[Reference]] = None*)

The semantics of a property or other elements that may have a semantic description is defined by a concept description.

The description of the concept should follow a standardized schema (realized as data specification template).

Constraint AASc-3a-004

For a *ConceptDescription* with `category` `PROPERTY` or `VALUE` using data specification IEC61360, the *DataSpecificationIEC61360.data_type* is mandatory and shall be one of: `DATE`, `STRING`, `STRING_TRANSLATABLE`, `INTEGER_MEASURE`, `INTEGER_COUNT`, `INTEGER_CURRENCY`, `REAL_MEASURE`, `REAL_COUNT`, `REAL_CURRENCY`, `BOOLEAN`, `RATIONAL`, `RATIONAL_MEASURE`, `TIME`, `TIMESTAMP`.

Note: Note: categories are deprecated since V3.0 of Part 1a of the document series "Details of the Asset Administration Shell".

Constraint AASc-3a-005

For a *ConceptDescription* with `category` `REFERENCE` using data specification template IEC61360, the *DataSpecificationIEC61360.data_type* shall be one of: `STRING`, `IRI`, `IRDI`.

Note: Note: categories are deprecated since V3.0 of Part 1a of the document series "Details of the Asset Administration Shell".

Constraint AASc-3a-006

For a *ConceptDescription* with `category` `DOCUMENT` using data specification IEC61360, the *DataSpecificationIEC61360.data_type* shall be one of `FILE`, `BLOB`, `HTML`

Note: Categories are deprecated since V3.0 of Part 1a of the document series "Details of the Asset Administration Shell".

Constraint AASc-3a-007

For a *ConceptDescription* with `category` `QUALIFIER_TYPE` using data specification IEC61360, the *DataSpecificationIEC61360.data_type* is mandatory and shall be defined.

Note: Categories are deprecated since V3.0 of Part 1a of the document series "Details of the Asset Administration Shell".

Constraint AASc-3a-008

For a *ConceptDescription* using data specification template IEC61360, *DataSpecificationIEC61360.definition* is mandatory and shall be defined at least in English.

Exception: The concept description describes a value, i.e. *DataSpecificationIEC61360.value* is defined.

Constraint AASc-3a-003
For a *ConceptDescription* using data specification template IEC61360, referenced via *DataSpecificationIEC61360.value_list* *ValueReferencePair.value_id* the *DataSpecificationIEC61360.value* shall be set.

over_is_case_of_or_empty() → Iterator[*Reference*]

Yield from *is_case_of* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*id: str*, *extensions: Optional[List[*Extension*]] = None*, *category: Optional[str] = None*, *id_short: Optional[str] = None*, *display_name: Optional[List[*LangStringNameType*]] = None*, *description: Optional[List[*LangStringTextType*]] = None*, *administration: Optional[*AdministrativeInformation*] = None*, *embedded_data_specifications: Optional[List[*EmbeddedDataSpecification*]] = None*, *is_case_of: Optional[List[*Reference*]] = None*) → None

Initialize with the given values.

is_case_of: Optional[List[*Reference*]]

Reference to an external definition the concept is compatible to or was derived from.

Note: It is recommended to use a global reference.

Note: Compare to is-case-of relationship in ISO 13584-32 & IEC EN 61360

---

class aas_core3.types.**ReferenceTypes**(*value*)

Reference types

EXTERNAL_REFERENCE = 'ExternalReference'

External reference.

MODEL_REFERENCE = 'ModelReference'

Model reference.

class aas_core3.types.**Reference**(*type:* ReferenceTypes, *keys: List[*Key*]*, *referred_semantic_id: Optional[*Reference*] = None*)

Reference to either a model element of the same or another AAS or to an external entity.

A reference is an ordered list of keys.

A model reference is an ordered list of keys, each key referencing an element. The complete list of keys may for example be concatenated to a path that then gives unique access to an element.

An external reference is a reference to an external entity.

Constraint AASd-121
For *Reference*'s the value of *Key.type* of the first key of " *keys* shall be one of `constants.GLOBALLY_IDENTIFIABLES`.

Constraint AASd-122
For external references, i.e. *Reference*'s with *type* = *ReferenceTypes.EXTERNAL_REFERENCE*, the value of *Key.type* of the first key of *keys* shall be one of `constants.GENERIC_GLOBALLY_IDENTIFIABLES`.

Constraint AASd-123
For model references, i.e. *Reference*'s with *type* = *ReferenceTypes.MODEL_REFERENCE*, the value of *Key.type* of the first key of *keys* shall be one of `constants.AAS_IDENTIFIABLES`.

Constraint AASd-124
For external references, i.e. *Reference*'s with *type* = *ReferenceTypes.EXTERNAL_REFERENCE*, the last key of *keys* shall be either one of `constants.GENERIC_GLOBALLY_IDENTIFIABLES` or one of `constants.GENERIC_FRAGMENT_KEYS`.

Constraint AASd-125
For model references, i.e. *Reference*'s with *type* = *ReferenceTypes.MODEL_REFERENCE*, with more than one key in *keys* the value of *Key.type* of each of the keys following the first key of *keys* shall be one of `constants.FRAGMENT_KEYS`.

Note: *Constraint AASd-125* ensures that the shortest path is used.

Constraint AASd-126
For model references, i.e. *Reference*'s with *type* = *ReferenceTypes.MODEL_REFERENCE*, with more than one key in *keys* the value of *Key.type* of the last key in the reference key chain may be one of `constants.GENERIC_FRAGMENT_KEYS` or no key at all shall have a value out of `constants.GENERIC_FRAGMENT_KEYS`.

Constraint AASd-127
For model references, i.e. *Reference*'s with *type* = *ReferenceTypes.MODEL_REFERENCE*, with more than one key in *keys* a key with *Key.type* *KeyTypes.FRAGMENT_REFERENCE* shall be preceded by a key with *Key.type* *KeyTypes.FILE* or *KeyTypes.BLOB*. All other AAS fragments, i.e. *Key.type* values out of `constants.AAS_SUBMODEL_ELEMENTS_AS_KEYS`, do not support fragments.

Note: Which kind of fragments are supported depends on the content type and the specification of allowed fragment identifiers for the corresponding resource being referenced via the reference.

Constraint AASd-128
For model references, i.e. *Reference*'s with *type* = *ReferenceTypes.MODEL_REFERENCE*, the *Key.value* of a *Key* preceded by a *Key* with *Key.type* = *KeyTypes.SUBMODEL_ELEMENT_LIST* is an integer number denoting the position in the array of the submodel element list.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*type:* ReferenceTypes, *keys: List[*Key*]*, *referred_semantic_id: Optional[*Reference*] = None*) → None

Initialize with the given values.

type: *ReferenceTypes*

Type of the reference.

Denotes, whether reference is an external reference or a model reference.

keys: **List[*Key*]**

Unique references in their name space.

referred_semantic_id: **Optional[*Reference*]**

HasSemantics.semantic_id of the referenced model element (*type* = *ReferenceTypes.MODEL_REFERENCE*).

For external references there typically is no semantic ID.

Note: It is recommended to use a external reference.

class aas_core3.types.**Key**(*type:* KeyTypes, *value: str*)

A key is a reference to an element by its ID.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*type:* KeyTypes, *value: str*) → None

Initialize with the given values.

type: *KeyTypes*

Denotes which kind of entity is referenced.

In case *type* = *KeyTypes.GLOBAL_REFERENCE*, the key represents a reference to a source that can be globally identified.

In case *type* = *KeyTypes.FRAGMENT_REFERENCE* the key represents a bookmark or a similar local identifier within its parent element as specified by the key that precedes this key.

In all other cases the key references a model element of the same or of another AAS. The name of the model element is explicitly listed.

value: str

The key value, for example an IRDI or an URI

class aas_core3.types.**KeyTypes**(*value*)

Enumeration of different key value types within a key.

ANNOTATED_RELATIONSHIP_ELEMENT = 'AnnotatedRelationshipElement'

ASSET_ADMINISTRATION_SHELL = 'AssetAdministrationShell'

BASIC_EVENT_ELEMENT = 'BasicEventElement'

BLOB = 'Blob'

CAPABILITY = 'Capability'

CONCEPT_DESCRIPTION = 'ConceptDescription'

DATA_ELEMENT = 'DataElement'

Data element.

Note: Data Element is abstract, *i.e.* if a key uses *DATA_ELEMENT* the reference may be a Property, a File etc.

ENTITY = 'Entity'

EVENT_ELEMENT = 'EventElement'

Event.

Note: *EventElement* is abstract.

FILE = 'File'

FRAGMENT_REFERENCE = 'FragmentReference'

Bookmark or a similar local identifier of a subordinate part of a primary resource

GLOBAL_REFERENCE = 'GlobalReference'

IDENTIFIABLE = 'Identifiable'

Identifiable.

Note: Identifiable is abstract, i.e. if a key uses "Identifiable" the reference may be an Asset Administration Shell, a Submodel or a Concept Description.

MULTI_LANGUAGE_PROPERTY = 'MultiLanguageProperty'

Property with a value that can be provided in multiple languages

OPERATION = 'Operation'

PROPERTY = 'Property'

RANGE = 'Range'

Range with min and max

REFERABLE = 'Referable'

REFERENCE_ELEMENT = 'ReferenceElement'

Reference

RELATIONSHIP_ELEMENT = 'RelationshipElement'

Relationship

SUBMODEL = 'Submodel'

SUBMODEL_ELEMENT = 'SubmodelElement'

Submodel Element

Note: Submodel Element is abstract, *i.e.* if a key uses *SUBMODEL_ELEMENT* the reference may be a *Property*, an *Operation* etc.

SUBMODEL_ELEMENT_COLLECTION = 'SubmodelElementCollection'

Struct of Submodel Elements

SUBMODEL_ELEMENT_LIST = 'SubmodelElementList'

List of Submodel Elements

class aas_core3.types.DataTypeDefXSD(*value*)

Enumeration listing all XSD anySimpleTypes

ANY_URI = 'xs:anyURI'

BASE_64_BINARY = 'xs:base64Binary'

BOOLEAN = 'xs:boolean'

BYTE = 'xs:byte'

DATE = 'xs:date'

DATE_TIME = 'xs:dateTime'

DECIMAL = 'xs:decimal'

DOUBLE = 'xs:double'

DURATION = 'xs:duration'

FLOAT = 'xs:float'

G_DAY = 'xs:gDay'

G_MONTH = 'xs:gMonth'

G_MONTH_DAY = 'xs:gMonthDay'

G_YEAR = 'xs:gYear'

G_YEAR_MONTH = 'xs:gYearMonth'

HEX_BINARY = 'xs:hexBinary'

INT = 'xs:int'

INTEGER = 'xs:integer'

LONG = 'xs:long'

NEGATIVE_INTEGER = 'xs:negativeInteger'

NON_NEGATIVE_INTEGER = 'xs:nonNegativeInteger'

NON_POSITIVE_INTEGER = 'xs:nonPositiveInteger'

POSITIVE_INTEGER = 'xs:positiveInteger'

SHORT = 'xs:short'

STRING = 'xs:string'

TIME = 'xs:time'

UNSIGNED_BYTE = 'xs:unsignedByte'

UNSIGNED_INT = 'xs:unsignedInt'

UNSIGNED_LONG = 'xs:unsignedLong'

UNSIGNED_SHORT = 'xs:unsignedShort'

class aas_core3.types.**AbstractLangString**(*language: str*, *text: str*)

Strings with language tags

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**LangStringNameType**(*language: str*, *text: str*)

String with length 128 maximum and minimum 1 characters and with language tags

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**LangStringTextType**(*language: str*, *text: str*)

String with length 1023 maximum and minimum 1 characters and with language tags

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**Environment**(*asset_administration_shells: Optional[List[*AssetAdministrationShell*]] = None*, *submodels: Optional[List[*Submodel*]] = None*, *concept_descriptions: Optional[List[*ConceptDescription*]] = None*)

Container for the sets of different identifiables.

Note: w.r.t. file exchange: There is exactly one environment independent on how many files the contained elements are split. If the file is split then there shall be no element with the same identifier in two different files.

---

over_asset_administration_shells_or_empty() → Iterator[*AssetAdministrationShell*]

Yield from *asset_administration_shells* if set.

over_submodels_or_empty() → Iterator[*Submodel*]

Yield from *submodels* if set.

over_concept_descriptions_or_empty() → Iterator[*ConceptDescription*]

Yield from *concept_descriptions* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*asset_administration_shells: Optional[List[*AssetAdministrationShell*]] = None*, *submodels: Optional[List[*Submodel*]] = None*, *concept_descriptions: Optional[List[*ConceptDescription*]] = None*) → None

Initialize with the given values.

asset_administration_shells: Optional[List[*AssetAdministrationShell*]]

Asset administration shell

submodels: Optional[List[*Submodel*]]

Submodel

concept_descriptions: Optional[List[*ConceptDescription*]]

Concept description

class aas_core3.types.**DataSpecificationContent**

Data specification content is part of a data specification template and defines which additional attributes shall be added to the element instance that references the data specification template and meta information about the template itself.

Constraint AASc-3a-050
If the *DataSpecificationIEC61360* is used for an element, the value of *HasDataSpecification.embedded_data_specifications* shall contain the global reference to the IRI of the corresponding data specification template https://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/3/0

class aas_core3.types.**EmbeddedDataSpecification**(*data_specification_content:* DataSpecificationContent, *data_specification: Optional[*Reference*] = None*)

Embed the content of a data specification.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*data_specification_content:* DataSpecificationContent, *data_specification: Optional[*Reference*] = None*) → None

Initialize with the given values.

data_specification_content: *DataSpecificationContent*

Actual content of the data specification

data_specification: **Optional[*Reference*]**

Reference to the data specification

class aas_core3.types.**DataTypeIEC61360**(*value*)

An enumeration.

DATE = 'DATE'

values containing a calendar date, conformant to ISO 8601:2004 Format yyyy-mm-dd Example from IEC 61360-1:2017: “1999-05-31” is the [DATE] representation of: “31 May 1999”.

STRING = 'STRING'

values consisting of sequence of characters but cannot be translated into other languages

STRING_TRANSLATABLE = 'STRING_TRANSLATABLE'

values containing string but shall be represented as different string in different languages

INTEGER_MEASURE = 'INTEGER_MEASURE'

values containing values that are measure of type INTEGER. In addition such a value comes with a physical unit.

INTEGER_COUNT = 'INTEGER_COUNT'

values containing values of type INTEGER but are no currencies or measures

INTEGER_CURRENCY = 'INTEGER_CURRENCY'

values containing values of type INTEGER that are currencies

REAL_MEASURE = 'REAL_MEASURE'

values containing values that are measures of type REAL. In addition such a value comes with a physical unit.

REAL_COUNT = 'REAL_COUNT'

values containing numbers that can be written as a terminating or non-terminating decimal; a rational or irrational number but are no currencies or measures

REAL_CURRENCY = 'REAL_CURRENCY'

values containing values of type REAL that are currencies

BOOLEAN = 'BOOLEAN'

values representing truth of logic or Boolean algebra (TRUE, FALSE)

IRI = 'IRI'

values containing values of type STRING conformant to Rfc 3987

Note: In IEC61360-1 (2017) only URI is supported. An IRI type allows in particular to express an URL or an URI.

IRDI = 'IRDI'

values conforming to ISO/IEC 11179 series global identifier sequences

IRDI can be used instead of the more specific data types ICID or ISO29002_IRDI.

ICID values are value conformant to an IRDI, where the delimiter between RAI and ID is "#" while the delimiter between DI and VI is confined to "##"

ISO29002_IRDI values are values containing a global identifier that identifies an administrated item in a registry. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5. The identifier shall fulfil the requirements specified in ISO/TS 29002-5 for an "international registration data identifier" (IRDI).

RATIONAL = 'RATIONAL'

values containing values of type rational

RATIONAL_MEASURE = 'RATIONAL_MEASURE'

values containing values of type rational. In addition such a value comes with a physical unit.

TIME = 'TIME'

values containing a time, conformant to ISO 8601:2004 but restricted to what is allowed in the corresponding type in xml.

Format hh:mm (ECLASS)

Example from IEC 61360-1:2017: "13:20:00-05:00" is the [TIME] representation of: 1.20 p.m. for Eastern Standard Time, which is 5 hours behind Coordinated Universal Time (UTC).

TIMESTAMP = 'TIMESTAMP'

values containing a time, conformant to ISO 8601:2004 but restricted to what is allowed in the corresponding type in xml.

Format yyyy-mm-dd hh:mm (ECLASS)

FILE = 'FILE'

values containing an address to a file. The values are of type URI and can represent an absolute or relative path.

Note: IEC61360 does not support the file type.

HTML = 'HTML'

Values containing string with any sequence of characters, using the syntax of HTML5 (see W3C Recommendation 28:2014)

BLOB = 'BLOB'

values containing the content of a file. Values may be binaries.

HTML conformant to HTML5 is a special blob.

In IEC61360 binary is for a sequence of bits, each bit being represented by "0" and "1" only. A binary is a blob but a blob may also contain other source code.

class aas_core3.types.**LevelType**(*min: bool*, *nom: bool*, *typ: bool*, *max: bool*)

Value represented by up to four variants of a numeric value in a specific role: `MIN`, `NOM`, `TYP` and `MAX`. True means that the value is available, false means the value is not available.

EXAMPLE from [IEC61360-1]: In the case of having a property which is of the LEVEL_TYPE min/max expressing a range only those two values need to be provided.

Note: This is how AAS deals with the following combinations of level types:

- Either all attributes are false. In this case the concept is mapped to a *Property* and level type is ignored.
- At most one of the attributes is set to true. In this case the concept is mapped to a *Property*.
- Min and max are set to true. In this case the concept is mapped to a *Range*.
- More than one attribute is set to true but not min and max only (see second case). In this case the concept is mapped to a *SubmodelElementCollection* with the corresponding number of Properties. Example: If attribute *min* and *nom* are set to true then the concept is mapped to a *SubmodelElementCollection* with two Properties within: min and nom. The data type of both Properties is the same.

Note: In the cases 2. and 4. the `Property.semantic_id` of the Property or Properties within the *SubmodelElementCollection* needs to include information about the level type. Otherwise, the semantics is not described in a unique way. Please refer to the specification.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*min: bool*, *nom: bool*, *typ: bool*, *max: bool*) → None

Initialize with the given values.

min: bool

Minimum of the value

nom: bool

Nominal value (value as designated)

typ: bool

Value as typically present

max: bool

Maximum of the value

class aas_core3.types.**ValueReferencePair**(*value: str*, *value_id:* Reference)

A value reference pair within a value list. Each value has a global unique id defining its semantic.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*value: str*, *value_id:* Reference) → None

Initialize with the given values.

value: str

The value of the referenced concept definition of the value in *value_id*.

value_id: *Reference*

Global unique id of the value.

Note: It is recommended to use a global reference.

---

class aas_core3.types.**ValueList**(*value_reference_pairs: List[*ValueReferencePair*]*)

A set of value reference pairs.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*value_reference_pairs: List[*ValueReferencePair*]*) → None

Initialize with the given values.

value_reference_pairs: List[*ValueReferencePair*]

A pair of a value together with its global unique id.

class aas_core3.types.**LangStringPreferredNameTypeIEC61360**(*language: str*, *text: str*)

String with length 255 maximum and minimum 1 characters and with language tags

---

Note: It is advised to keep the length of the name limited to 35 characters.

---

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**LangStringShortNameTypeIEC61360**(*language: str*, *text: str*)

String with length 18 maximum and minimum 1 characters and with language tags

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**LangStringDefinitionTypeIEC61360**(*language: str*, *text: str*)

String with length 1023 maximum and minimum 1 characters and with language tags

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield

instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield

instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*language: str*, *text: str*) → None

Initialize with the given values.

language: str

Language tag conforming to BCP 47

text: str

Text in the *language*

class aas_core3.types.**DataSpecificationIEC61360**(*preferred_name: List[LangStringPreferredNameTypeIEC61360]*, *short_name: Optional[List[LangStringShortNameTypeIEC61360]] = None*, *unit: Optional[str] = None*, *unit_id: Optional[Reference] = None*, *source_of_definition: Optional[str] = None*, *symbol: Optional[str] = None*, *data_type: Optional[DataTypeIEC61360] = None*, *definition: Optional[List[LangStringDefinitionTypeIEC61360]] = None*, *value_format: Optional[str] = None*, *value_list: Optional[ValueList] = None*, *value: Optional[str] = None*, *level_type: Optional[LevelType] = None*)

Content of data specification template for concept descriptions for properties, values and value lists conformant to IEC 61360.

Note: IEC61360 requires also a globally unique identifier for a concept description. This ID is not part of the data specification template. Instead the `ConceptDescription.id` as inherited via *Identifiable* is used. Same holds for administrative information like the version and revision.

Note: `ConceptDescription.id_short` and *short_name* are very similar. However, in this case the decision was to add *short_name* explicitly to the data specification. Same holds for `ConceptDescription.display_name` and *preferred_name*. Same holds for `ConceptDescription.description` and *definition*.

Constraint AASc-3a-010

If *value* is not empty then *value_list* shall be empty and vice versa.

Note: It is also possible that both *value* and *value_list* are empty. This is the case for concept descriptions that define the semantics of a property but do not have an enumeration (*value_list*) as data type.

Note: Although it is possible to define a *ConceptDescription* for a :attr:´value_list`, it is not possible to reuse this *value_list*. It is only possible to directly add a *value_list* as data type to a specific semantic definition of a property.

Constraint AASc-3a-009

If *data_type* one of: *DataTypeIEC61360.INTEGER_MEASURE*, *DataTypeIEC61360.REAL_MEASURE*, *DataTypeIEC61360.RATIONAL_MEASURE*, *DataTypeIEC61360.INTEGER_CURRENCY*, *DataTypeIEC61360.REAL_CURRENCY*, then *unit* or *unit_id* shall be defined.

over_short_name_or_empty() → Iterator[*LangStringShortNameTypeIEC61360*]

Yield from *short_name* if set.

over_definition_or_empty() → Iterator[*LangStringDefinitionTypeIEC61360*]

Yield from *definition* if set.

descend_once() → Iterator[*Class*]

Iterate over the instances referenced from this instance.

We do not recurse into the referenced instance.

Yield
instances directly referenced from this instance

descend() → Iterator[*Class*]

Iterate recursively over the instances referenced from this one.

Yield
instances recursively referenced from this instance

accept(*visitor:* AbstractVisitor) → None

Dispatch the `visitor` on this instance.

accept_with_context(*visitor:* AbstractVisitorWithContext*[ContextT]*, *context: ContextT*) → None

Dispatch the `visitor` on this instance in `context`.

transform(*transformer:* AbstractTransformer*[T]*) → T

Dispatch the `transformer` on this instance.

transform_with_context(*transformer:* AbstractTransformerWithContext*[ContextT, T]*, *context: ContextT*) → T

Dispatch the `transformer` on this instance in `context`.

__init__(*preferred_name: List[*LangStringPreferredNameTypeIEC61360*]*, *short_name: Optional[List[*LangStringShortNameTypeIEC61360*]] = None*, *unit: Optional[str] = None*, *unit_id: Optional[*Reference*] = None*, *source_of_definition: Optional[str] = None*, *symbol: Optional[str] = None*, *data_type: Optional[*DataTypeIEC61360*] = None*, *definition: Optional[List[*LangStringDefinitionTypeIEC61360*]] = None*, *value_format: Optional[str] = None*, *value_list: Optional[*ValueList*] = None*, *value: Optional[str] = None*, *level_type: Optional[*LevelType*] = None*) → None

Initialize with the given values.

preferred_name: List[*LangStringPreferredNameTypeIEC61360*]

Preferred name

Note: It is advised to keep the length of the name limited to 35 characters.

Constraint AASc-3a-002
preferred_name shall be provided at least in English.

short_name: Optional[List[*LangStringShortNameTypeIEC61360*]]

Short name

unit: Optional[str]

Unit

unit_id: Optional[*Reference*]

Unique unit id

unit and *unit_id* need to be consistent if both attributes are set

Note: It is recommended to use an external reference ID.

source_of_definition: Optional[str]
Source of definition

symbol: Optional[str]
Symbol

data_type: Optional[*DataTypeIEC61360*]
Data Type

definition: Optional[List[*LangStringDefinitionTypeIEC61360*]]
Definition in different languages

value_format: Optional[str]
Value Format

Note: The value format is based on ISO 13584-42 and IEC 61360-2.

value_list: Optional[*ValueList*]
List of allowed values

value: Optional[str]
Value

level_type: Optional[*LevelType*]
Set of levels.

class aas_core3.types.**AbstractVisitor**
Visit the instances of the model.

visit(*that:* Class) → None
Double-dispatch on `that`.

abstract visit_extension(*that:* Extension) → None
Visit `that`.

abstract visit_administrative_information(*that:* AdministrativeInformation) → None
Visit `that`.

abstract visit_qualifier(*that:* Qualifier) → None
Visit `that`.

abstract visit_asset_administration_shell(*that:* AssetAdministrationShell) → None
Visit `that`.

abstract visit_asset_information(*that:* AssetInformation) → None
Visit `that`.

abstract visit_resource(*that:* Resource) → None
Visit `that`.

abstract visit_specific_asset_id(*that:* SpecificAssetID) → None
Visit `that`.

abstract visit_submodel(*that:* Submodel) → None
Visit that.

abstract visit_relationship_element(*that:* RelationshipElement) → None
Visit that.

abstract visit_submodel_element_list(*that:* SubmodelElementList) → None
Visit that.

abstract visit_submodel_element_collection(*that:* SubmodelElementCollection) → None
Visit that.

abstract visit_property(*that:* Property) → None
Visit that.

abstract visit_multi_language_property(*that:* MultiLanguageProperty) → None
Visit that.

abstract visit_range(*that:* Range) → None
Visit that.

abstract visit_reference_element(*that:* ReferenceElement) → None
Visit that.

abstract visit_blob(*that:* Blob) → None
Visit that.

abstract visit_file(*that:* File) → None
Visit that.

abstract visit_annotated_relationship_element(*that:* AnnotatedRelationshipElement) → None
Visit that.

abstract visit_entity(*that:* Entity) → None
Visit that.

abstract visit_event_payload(*that:* EventPayload) → None
Visit that.

abstract visit_basic_event_element(*that:* BasicEventElement) → None
Visit that.

abstract visit_operation(*that:* Operation) → None
Visit that.

abstract visit_operation_variable(*that:* OperationVariable) → None
Visit that.

abstract visit_capability(*that:* Capability) → None
Visit that.

abstract visit_concept_description(*that:* ConceptDescription) → None
Visit that.

abstract visit_reference(*that:* Reference) → None
Visit that.

abstract visit_key(*that:* Key) → None
Visit that.

abstract visit_lang_string_name_type(*that:* LangStringNameType) → None

Visit `that`.

abstract visit_lang_string_text_type(*that:* LangStringTextType) → None

Visit `that`.

abstract visit_environment(*that:* Environment) → None

Visit `that`.

abstract visit_embedded_data_specification(*that:* EmbeddedDataSpecification) → None

Visit `that`.

abstract visit_level_type(*that:* LevelType) → None

Visit `that`.

abstract visit_value_reference_pair(*that:* ValueReferencePair) → None

Visit `that`.

abstract visit_value_list(*that:* ValueList) → None

Visit `that`.

abstract visit_lang_string_preferred_name_type_iec_61360(*that:* LangStringPreferredNameTypeIEC61360) → None

Visit `that`.

abstract visit_lang_string_short_name_type_iec_61360(*that:* LangStringShortNameTypeIEC61360) → None

Visit `that`.

abstract visit_lang_string_definition_type_iec_61360(*that:* LangStringDefinitionTypeIEC61360) → None

Visit `that`.

abstract visit_data_specification_iec_61360(*that:* DataSpecificationIEC61360) → None

Visit `that`.

class aas_core3.types.**AbstractVisitorWithContext**(**args*, ***kwds*)

Visit the instances of the model with context.

visit_with_context(*that:* Class, *context: ContextT*) → None

Double-dispatch on `that`.

abstract visit_extension_with_context(*that:* Extension, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_administrative_information_with_context(*that:* AdministrativeInformation, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_qualifier_with_context(*that:* Qualifier, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_asset_administration_shell_with_context(*that:* AssetAdministrationShell, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_asset_information_with_context(*that:* AssetInformation, *context: ContextT*) → None

Visit that in context.

abstract visit_resource_with_context(*that:* Resource, *context: ContextT*) → None

Visit that in context.

abstract visit_specific_asset_id_with_context(*that:* SpecificAssetID, *context: ContextT*) → None

Visit that in context.

abstract visit_submodel_with_context(*that:* Submodel, *context: ContextT*) → None

Visit that in context.

abstract visit_relationship_element_with_context(*that:* RelationshipElement, *context: ContextT*) → None

Visit that in context.

abstract visit_submodel_element_list_with_context(*that:* SubmodelElementList, *context: ContextT*) → None

Visit that in context.

abstract visit_submodel_element_collection_with_context(*that:* SubmodelElementCollection, *context: ContextT*) → None

Visit that in context.

abstract visit_property_with_context(*that:* Property, *context: ContextT*) → None

Visit that in context.

abstract visit_multi_language_property_with_context(*that:* MultiLanguageProperty, *context: ContextT*) → None

Visit that in context.

abstract visit_range_with_context(*that:* Range, *context: ContextT*) → None

Visit that in context.

abstract visit_reference_element_with_context(*that:* ReferenceElement, *context: ContextT*) → None

Visit that in context.

abstract visit_blob_with_context(*that:* Blob, *context: ContextT*) → None

Visit that in context.

abstract visit_file_with_context(*that:* File, *context: ContextT*) → None

Visit that in context.

abstract visit_annotated_relationship_element_with_context(*that:* AnnotatedRelationshipElement, *context: ContextT*) → None

Visit that in context.

abstract visit_entity_with_context(*that:* Entity, *context: ContextT*) → None

Visit that in context.

abstract visit_event_payload_with_context(*that:* EventPayload, *context: ContextT*) → None

Visit that in context.

abstract visit_basic_event_element_with_context(*that:* BasicEventElement, *context: ContextT*) → None

Visit that in context.

abstract visit_operation_with_context(*that:* Operation, *context: ContextT*) → None

Visit that in context.

abstract visit_operation_variable_with_context(*that:* OperationVariable, *context: ContextT*) → None

Visit that in context.

abstract visit_capability_with_context(*that:* Capability, *context: ContextT*) → None

Visit that in context.

abstract visit_concept_description_with_context(*that:* ConceptDescription, *context: ContextT*) → None

Visit that in context.

abstract visit_reference_with_context(*that:* Reference, *context: ContextT*) → None

Visit that in context.

abstract visit_key_with_context(*that:* Key, *context: ContextT*) → None

Visit that in context.

abstract visit_lang_string_name_type_with_context(*that:* LangStringNameType, *context: ContextT*) → None

Visit that in context.

abstract visit_lang_string_text_type_with_context(*that:* LangStringTextType, *context: ContextT*) → None

Visit that in context.

abstract visit_environment_with_context(*that:* Environment, *context: ContextT*) → None

Visit that in context.

abstract visit_embedded_data_specification_with_context(*that:* EmbeddedDataSpecification, *context: ContextT*) → None

Visit that in context.

abstract visit_level_type_with_context(*that:* LevelType, *context: ContextT*) → None

Visit that in context.

abstract visit_value_reference_pair_with_context(*that:* ValueReferencePair, *context: ContextT*) → None

Visit that in context.

abstract visit_value_list_with_context(*that:* ValueList, *context: ContextT*) → None

Visit that in context.

abstract visit_lang_string_preferred_name_type_iec_61360_with_context(*that:* LangStringPreferredNameTypeIEC61360, *context: ContextT*) → None

Visit that in context.

abstract visit_lang_string_short_name_type_iec_61360_with_context(*that:* LangStringShortNameTypeIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_lang_string_definition_type_iec_61360_with_context(*that:* LangStringDefinitionTypeIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

abstract visit_data_specification_iec_61360_with_context(*that:* DataSpecificationIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

__orig_bases__ = (typing.Generic[~ContextT],)

__parameters__ = (~ContextT,)

class aas_core3.types.**PassThroughVisitor**

Visit the instances of the model without action.

This visitor is not meant to be directly used. Instead, you usually inherit from it, and implement only the relevant visit methods.

visit(*that:* Class) → None

Double-dispatch on `that`.

visit_extension(*that:* Extension) → None

Visit `that`.

visit_administrative_information(*that:* AdministrativeInformation) → None

Visit `that`.

visit_qualifier(*that:* Qualifier) → None

Visit `that`.

visit_asset_administration_shell(*that:* AssetAdministrationShell) → None

Visit `that`.

visit_asset_information(*that:* AssetInformation) → None

Visit `that`.

visit_resource(*that:* Resource) → None

Visit `that`.

visit_specific_asset_id(*that:* SpecificAssetID) → None

Visit `that`.

visit_submodel(*that:* Submodel) → None

Visit `that`.

visit_relationship_element(*that:* RelationshipElement) → None

Visit `that`.

visit_submodel_element_list(*that:* SubmodelElementList) → None

Visit `that`.

visit_submodel_element_collection(*that:* SubmodelElementCollection) → None
Visit that.

visit_property(*that:* Property) → None
Visit that.

visit_multi_language_property(*that:* MultiLanguageProperty) → None
Visit that.

visit_range(*that:* Range) → None
Visit that.

visit_reference_element(*that:* ReferenceElement) → None
Visit that.

visit_blob(*that:* Blob) → None
Visit that.

visit_file(*that:* File) → None
Visit that.

visit_annotated_relationship_element(*that:* AnnotatedRelationshipElement) → None
Visit that.

visit_entity(*that:* Entity) → None
Visit that.

visit_event_payload(*that:* EventPayload) → None
Visit that.

visit_basic_event_element(*that:* BasicEventElement) → None
Visit that.

visit_operation(*that:* Operation) → None
Visit that.

visit_operation_variable(*that:* OperationVariable) → None
Visit that.

visit_capability(*that:* Capability) → None
Visit that.

visit_concept_description(*that:* ConceptDescription) → None
Visit that.

visit_reference(*that:* Reference) → None
Visit that.

visit_key(*that:* Key) → None
Visit that.

visit_lang_string_name_type(*that:* LangStringNameType) → None
Visit that.

visit_lang_string_text_type(*that:* LangStringTextType) → None
Visit that.

visit_environment(*that:* Environment) → None
Visit that.

visit_embedded_data_specification(*that:* EmbeddedDataSpecification) → None

Visit `that`.

visit_level_type(*that:* LevelType) → None

Visit `that`.

visit_value_reference_pair(*that:* ValueReferencePair) → None

Visit `that`.

visit_value_list(*that:* ValueList) → None

Visit `that`.

visit_lang_string_preferred_name_type_iec_61360(*that:* LangStringPreferredNameTypeIEC61360) → None

Visit `that`.

visit_lang_string_short_name_type_iec_61360(*that:* LangStringShortNameTypeIEC61360) → None

Visit `that`.

visit_lang_string_definition_type_iec_61360(*that:* LangStringDefinitionTypeIEC61360) → None

Visit `that`.

visit_data_specification_iec_61360(*that:* DataSpecificationIEC61360) → None

Visit `that`.

class aas_core3.types.**PassThroughVisitorWithContext**(**args*, ***kwds*)

Visit the instances of the model without action and in context.

This visitor is not meant to be directly used. Instead, you usually inherit from it, and implement only the relevant visit methods.

visit_with_context(*that:* Class, *context: ContextT*) → None

Double-dispatch on `that`.

visit_extension_with_context(*that:* Extension, *context: ContextT*) → None

Visit `that` in `context`.

visit_administrative_information_with_context(*that:* AdministrativeInformation, *context: ContextT*) → None

Visit `that` in `context`.

visit_qualifier_with_context(*that:* Qualifier, *context: ContextT*) → None

Visit `that` in `context`.

visit_asset_administration_shell_with_context(*that:* AssetAdministrationShell, *context: ContextT*) → None

Visit `that` in `context`.

visit_asset_information_with_context(*that:* AssetInformation, *context: ContextT*) → None

Visit `that` in `context`.

visit_resource_with_context(*that:* Resource, *context: ContextT*) → None

Visit `that` in `context`.

visit_specific_asset_id_with_context(*that:* SpecificAssetID, *context: ContextT*) → None

Visit `that` in `context`.

visit_submodel_with_context(*that:* Submodel, *context: ContextT*) → None

Visit that in context.

visit_relationship_element_with_context(*that:* RelationshipElement, *context: ContextT*) → None

Visit that in context.

visit_submodel_element_list_with_context(*that:* SubmodelElementList, *context: ContextT*) → None

Visit that in context.

visit_submodel_element_collection_with_context(*that:* SubmodelElementCollection, *context: ContextT*) → None

Visit that in context.

visit_property_with_context(*that:* Property, *context: ContextT*) → None

Visit that in context.

visit_multi_language_property_with_context(*that:* MultiLanguageProperty, *context: ContextT*) → None

Visit that in context.

visit_range_with_context(*that:* Range, *context: ContextT*) → None

Visit that in context.

visit_reference_element_with_context(*that:* ReferenceElement, *context: ContextT*) → None

Visit that in context.

visit_blob_with_context(*that:* Blob, *context: ContextT*) → None

Visit that in context.

visit_file_with_context(*that:* File, *context: ContextT*) → None

Visit that in context.

visit_annotated_relationship_element_with_context(*that:* AnnotatedRelationshipElement, *context: ContextT*) → None

Visit that in context.

visit_entity_with_context(*that:* Entity, *context: ContextT*) → None

Visit that in context.

visit_event_payload_with_context(*that:* EventPayload, *context: ContextT*) → None

Visit that in context.

visit_basic_event_element_with_context(*that:* BasicEventElement, *context: ContextT*) → None

Visit that in context.

visit_operation_with_context(*that:* Operation, *context: ContextT*) → None

Visit that in context.

visit_operation_variable_with_context(*that:* OperationVariable, *context: ContextT*) → None

Visit that in context.

visit_capability_with_context(*that:* Capability, *context: ContextT*) → None

Visit that in context.

visit_concept_description_with_context(*that:* ConceptDescription, *context: ContextT*) → None

Visit that in context.

visit_reference_with_context(*that:* Reference, *context: ContextT*) → None

Visit `that` in `context`.

visit_key_with_context(*that:* Key, *context: ContextT*) → None

Visit `that` in `context`.

visit_lang_string_name_type_with_context(*that:* LangStringNameType, *context: ContextT*) → None

Visit `that` in `context`.

visit_lang_string_text_type_with_context(*that:* LangStringTextType, *context: ContextT*) → None

Visit `that` in `context`.

visit_environment_with_context(*that:* Environment, *context: ContextT*) → None

Visit `that` in `context`.

visit_embedded_data_specification_with_context(*that:* EmbeddedDataSpecification, *context: ContextT*) → None

Visit `that` in `context`.

visit_level_type_with_context(*that:* LevelType, *context: ContextT*) → None

Visit `that` in `context`.

visit_value_reference_pair_with_context(*that:* ValueReferencePair, *context: ContextT*) → None

Visit `that` in `context`.

visit_value_list_with_context(*that:* ValueList, *context: ContextT*) → None

Visit `that` in `context`.

visit_lang_string_preferred_name_type_iec_61360_with_context(*that:* LangStringPreferredNameTypeIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

visit_lang_string_short_name_type_iec_61360_with_context(*that:* LangStringShortNameTypeIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

visit_lang_string_definition_type_iec_61360_with_context(*that:* LangStringDefinitionTypeIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

visit_data_specification_iec_61360_with_context(*that:* DataSpecificationIEC61360, *context: ContextT*) → None

Visit `that` in `context`.

__orig_bases__ = (aas_core3.types.AbstractVisitorWithContext[~ContextT],)

__parameters__ = (~ContextT,)

class aas_core3.types.**AbstractTransformer**(**args*, ***kwds*)

Transform the instances of the model.

transform(*that:* Class) → T

Double-dispatch on `that`.

abstract transform_extension(*that:* Extension) → T
Transform that.

abstract transform_administrative_information(*that:* AdministrativeInformation) → T
Transform that.

abstract transform_qualifier(*that:* Qualifier) → T
Transform that.

abstract transform_asset_administration_shell(*that:* AssetAdministrationShell) → T
Transform that.

abstract transform_asset_information(*that:* AssetInformation) → T
Transform that.

abstract transform_resource(*that:* Resource) → T
Transform that.

abstract transform_specific_asset_id(*that:* SpecificAssetID) → T
Transform that.

abstract transform_submodel(*that:* Submodel) → T
Transform that.

abstract transform_relationship_element(*that:* RelationshipElement) → T
Transform that.

abstract transform_submodel_element_list(*that:* SubmodelElementList) → T
Transform that.

abstract transform_submodel_element_collection(*that:* SubmodelElementCollection) → T
Transform that.

abstract transform_property(*that:* Property) → T
Transform that.

abstract transform_multi_language_property(*that:* MultiLanguageProperty) → T
Transform that.

abstract transform_range(*that:* Range) → T
Transform that.

abstract transform_reference_element(*that:* ReferenceElement) → T
Transform that.

abstract transform_blob(*that:* Blob) → T
Transform that.

abstract transform_file(*that:* File) → T
Transform that.

abstract transform_annotated_relationship_element(*that:* AnnotatedRelationshipElement) → T
Transform that.

abstract transform_entity(*that:* Entity) → T
Transform that.

abstract transform_event_payload(*that:* EventPayload) → T
Transform that.

abstract transform_basic_event_element(*that:* BasicEventElement) → T

Transform that.

abstract transform_operation(*that:* Operation) → T

Transform that.

abstract transform_operation_variable(*that:* OperationVariable) → T

Transform that.

abstract transform_capability(*that:* Capability) → T

Transform that.

abstract transform_concept_description(*that:* ConceptDescription) → T

Transform that.

abstract transform_reference(*that:* Reference) → T

Transform that.

abstract transform_key(*that:* Key) → T

Transform that.

abstract transform_lang_string_name_type(*that:* LangStringNameType) → T

Transform that.

abstract transform_lang_string_text_type(*that:* LangStringTextType) → T

Transform that.

abstract transform_environment(*that:* Environment) → T

Transform that.

abstract transform_embedded_data_specification(*that:* EmbeddedDataSpecification) → T

Transform that.

abstract transform_level_type(*that:* LevelType) → T

Transform that.

abstract transform_value_reference_pair(*that:* ValueReferencePair) → T

Transform that.

abstract transform_value_list(*that:* ValueList) → T

Transform that.

abstract transform_lang_string_preferred_name_type_iec_61360(*that:* LangStringPreferredNameTypeIEC61360) → T

Transform that.

abstract transform_lang_string_short_name_type_iec_61360(*that:* LangStringShortNameTypeIEC61360) → T

Transform that.

abstract transform_lang_string_definition_type_iec_61360(*that:* LangStringDefinitionTypeIEC61360) → T

Transform that.

abstract transform_data_specification_iec_61360(*that:* DataSpecificationIEC61360) → T

Transform that.

__orig_bases__ = (typing.Generic[~T],)

__parameters__ = (~T,)

class aas_core3.types.**AbstractTransformerWithContext**(**args*, ***kwds*)

Transform the instances of the model in context.

transform_with_context(*that:* Class, *context: ContextT*) → T

Double-dispatch on that.

abstract transform_extension_with_context(*that:* Extension, *context: ContextT*) → T

Transform that in context.

abstract transform_administrative_information_with_context(*that:* AdministrativeInformation, *context: ContextT*) → T

Transform that in context.

abstract transform_qualifier_with_context(*that:* Qualifier, *context: ContextT*) → T

Transform that in context.

abstract transform_asset_administration_shell_with_context(*that:* AssetAdministrationShell, *context: ContextT*) → T

Transform that in context.

abstract transform_asset_information_with_context(*that:* AssetInformation, *context: ContextT*) → T

Transform that in context.

__orig_bases__ = (typing.Generic[~ContextT, ~T],)

__parameters__ = (~ContextT, ~T)

abstract transform_resource_with_context(*that:* Resource, *context: ContextT*) → T

Transform that in context.

abstract transform_specific_asset_id_with_context(*that:* SpecificAssetID, *context: ContextT*) → T

Transform that in context.

abstract transform_submodel_with_context(*that:* Submodel, *context: ContextT*) → T

Transform that in context.

abstract transform_relationship_element_with_context(*that:* RelationshipElement, *context: ContextT*) → T

Transform that in context.

abstract transform_submodel_element_list_with_context(*that:* SubmodelElementList, *context: ContextT*) → T

Transform that in context.

abstract transform_submodel_element_collection_with_context(*that:* SubmodelElementCollection, *context: ContextT*) → T

Transform that in context.

abstract transform_property_with_context(*that:* Property, *context: ContextT*) → T

Transform that in context.

abstract transform_multi_language_property_with_context(*that:* MultiLanguageProperty, *context: ContextT*) → T

Transform that in context.

abstract transform_range_with_context(*that:* Range, *context: ContextT*) → T

Transform that in context.

abstract transform_reference_element_with_context(*that:* ReferenceElement, *context: ContextT*) → T

Transform that in context.

abstract transform_blob_with_context(*that:* Blob, *context: ContextT*) → T

Transform that in context.

abstract transform_file_with_context(*that:* File, *context: ContextT*) → T

Transform that in context.

abstract transform_annotated_relationship_element_with_context(*that:* AnnotatedRelationshipElement, *context: ContextT*) → T

Transform that in context.

abstract transform_entity_with_context(*that:* Entity, *context: ContextT*) → T

Transform that in context.

abstract transform_event_payload_with_context(*that:* EventPayload, *context: ContextT*) → T

Transform that in context.

abstract transform_basic_event_element_with_context(*that:* BasicEventElement, *context: ContextT*) → T

Transform that in context.

abstract transform_operation_with_context(*that:* Operation, *context: ContextT*) → T

Transform that in context.

abstract transform_operation_variable_with_context(*that:* OperationVariable, *context: ContextT*) → T

Transform that in context.

abstract transform_capability_with_context(*that:* Capability, *context: ContextT*) → T

Transform that in context.

abstract transform_concept_description_with_context(*that:* ConceptDescription, *context: ContextT*) → T

Transform that in context.

abstract transform_reference_with_context(*that:* Reference, *context: ContextT*) → T

Transform that in context.

abstract transform_key_with_context(*that:* Key, *context: ContextT*) → T

Transform that in context.

abstract transform_lang_string_name_type_with_context(*that:* LangStringNameType, *context: ContextT*) → T

Transform that in context.

abstract transform_lang_string_text_type_with_context(*that:* LangStringTextType, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_environment_with_context(*that:* Environment, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_embedded_data_specification_with_context(*that:* EmbeddedDataSpecification, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_level_type_with_context(*that:* LevelType, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_value_reference_pair_with_context(*that:* ValueReferencePair, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_value_list_with_context(*that:* ValueList, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_lang_string_preferred_name_type_iec_61360_with_context(*that:* LangStringPreferredNameTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_lang_string_short_name_type_iec_61360_with_context(*that:* LangStringShortNameTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_lang_string_definition_type_iec_61360_with_context(*that:* LangStringDefinitionTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

abstract transform_data_specification_iec_61360_with_context(*that:* DataSpecificationIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

class aas_core3.types.**TransformerWithDefault**(*default: T*)

Transform the instances of the model.

If you do not override the transformation methods, they simply return *default*.

__orig_bases__ = (aas_core3.types.AbstractTransformer[~T],)

__parameters__ = (~T,)

__init__(*default: T*) → None

Initialize with the given `default` value.

default: T

Default value which is returned if no override of the transformation

transform(*that:* Class) → T

Double-dispatch on `that`.

transform_extension(*that:* Extension) → T

Transform `that`.

transform_administrative_information(*that:* AdministrativeInformation) → T

Transform `that`.

transform_qualifier(*that:* Qualifier) → T

Transform `that`.

transform_asset_administration_shell(*that:* AssetAdministrationShell) → T

Transform `that`.

transform_asset_information(*that:* AssetInformation) → T

Transform `that`.

transform_resource(*that:* Resource) → T

Transform `that`.

transform_specific_asset_id(*that:* SpecificAssetID) → T

Transform `that`.

transform_submodel(*that:* Submodel) → T

Transform `that`.

transform_relationship_element(*that:* RelationshipElement) → T

Transform `that`.

transform_submodel_element_list(*that:* SubmodelElementList) → T

Transform `that`.

transform_submodel_element_collection(*that:* SubmodelElementCollection) → T

Transform `that`.

transform_property(*that:* Property) → T

Transform `that`.

transform_multi_language_property(*that:* MultiLanguageProperty) → T

Transform `that`.

transform_range(*that:* Range) → T

Transform `that`.

transform_reference_element(*that:* ReferenceElement) → T

Transform `that`.

transform_blob(*that:* Blob) → T

Transform that.

transform_file(*that:* File) → T

Transform that.

transform_annotated_relationship_element(*that:* AnnotatedRelationshipElement) → T

Transform that.

transform_entity(*that:* Entity) → T

Transform that.

transform_event_payload(*that:* EventPayload) → T

Transform that.

transform_basic_event_element(*that:* BasicEventElement) → T

Transform that.

transform_operation(*that:* Operation) → T

Transform that.

transform_operation_variable(*that:* OperationVariable) → T

Transform that.

transform_capability(*that:* Capability) → T

Transform that.

transform_concept_description(*that:* ConceptDescription) → T

Transform that.

transform_reference(*that:* Reference) → T

Transform that.

transform_key(*that:* Key) → T

Transform that.

transform_lang_string_name_type(*that:* LangStringNameType) → T

Transform that.

transform_lang_string_text_type(*that:* LangStringTextType) → T

Transform that.

transform_environment(*that:* Environment) → T

Transform that.

transform_embedded_data_specification(*that:* EmbeddedDataSpecification) → T

Transform that.

transform_level_type(*that:* LevelType) → T

Transform that.

transform_value_reference_pair(*that:* ValueReferencePair) → T

Transform that.

transform_value_list(*that:* ValueList) → T

Transform that.

transform_lang_string_preferred_name_type_iec_61360(*that:* LangStringPreferredNameTypeIEC61360) → T

Transform `that`.

transform_lang_string_short_name_type_iec_61360(*that:* LangStringShortNameTypeIEC61360) → T

Transform `that`.

transform_lang_string_definition_type_iec_61360(*that:* LangStringDefinitionTypeIEC61360) → T

Transform `that`.

transform_data_specification_iec_61360(*that:* DataSpecificationIEC61360) → T

Transform `that`.

class aas_core3.types.**TransformerWithDefaultAndContext**(*default: T*)

Transform the instances of the model in context.

If you do not override the transformation methods, they simply return *default*.

__orig_bases__ = (aas_core3.types.AbstractTransformerWithContext[~ContextT, ~T],)

__parameters__ = (~ContextT, ~T)

__init__(*default: T*) → None

Initialize with the given `default` value.

default: T

Default value which is returned if no override of the transformation

transform_with_context(*that:* Class, *context: ContextT*) → T

Double-dispatch on `that`.

transform_extension_with_context(*that:* Extension, *context: ContextT*) → T

Transform `that` in `context`.

transform_administrative_information_with_context(*that:* AdministrativeInformation, *context: ContextT*) → T

Transform `that` in `context`.

transform_qualifier_with_context(*that:* Qualifier, *context: ContextT*) → T

Transform `that` in `context`.

transform_asset_administration_shell_with_context(*that:* AssetAdministrationShell, *context: ContextT*) → T

Transform `that` in `context`.

transform_asset_information_with_context(*that:* AssetInformation, *context: ContextT*) → T

Transform `that` in `context`.

transform_resource_with_context(*that:* Resource, *context: ContextT*) → T

Transform `that` in `context`.

transform_specific_asset_id_with_context(*that:* SpecificAssetID, *context: ContextT*) → T

Transform `that` in `context`.

transform_submodel_with_context(*that:* Submodel, *context: ContextT*) → T

Transform `that` in `context`.

transform_relationship_element_with_context(*that:* RelationshipElement, *context: ContextT*) → T

Transform `that` in `context`.

transform_submodel_element_list_with_context(*that:* SubmodelElementList, *context: ContextT*) → T

Transform `that` in `context`.

transform_submodel_element_collection_with_context(*that:* SubmodelElementCollection, *context: ContextT*) → T

Transform `that` in `context`.

transform_property_with_context(*that:* Property, *context: ContextT*) → T

Transform `that` in `context`.

transform_multi_language_property_with_context(*that:* MultiLanguageProperty, *context: ContextT*) → T

Transform `that` in `context`.

transform_range_with_context(*that:* Range, *context: ContextT*) → T

Transform `that` in `context`.

transform_reference_element_with_context(*that:* ReferenceElement, *context: ContextT*) → T

Transform `that` in `context`.

transform_blob_with_context(*that:* Blob, *context: ContextT*) → T

Transform `that` in `context`.

transform_file_with_context(*that:* File, *context: ContextT*) → T

Transform `that` in `context`.

transform_annotated_relationship_element_with_context(*that:* AnnotatedRelationshipElement, *context: ContextT*) → T

Transform `that` in `context`.

transform_entity_with_context(*that:* Entity, *context: ContextT*) → T

Transform `that` in `context`.

transform_event_payload_with_context(*that:* EventPayload, *context: ContextT*) → T

Transform `that` in `context`.

transform_basic_event_element_with_context(*that:* BasicEventElement, *context: ContextT*) → T

Transform `that` in `context`.

transform_operation_with_context(*that:* Operation, *context: ContextT*) → T

Transform `that` in `context`.

transform_operation_variable_with_context(*that:* OperationVariable, *context: ContextT*) → T

Transform `that` in `context`.

transform_capability_with_context(*that:* Capability, *context: ContextT*) → T

Transform `that` in `context`.

transform_concept_description_with_context(*that:* ConceptDescription, *context: ContextT*) → T

Transform `that` in `context`.

transform_reference_with_context(*that:* Reference, *context: ContextT*) → T

Transform `that` in `context`.

transform_key_with_context(*that:* Key, *context: ContextT*) → T

Transform `that` in `context`.

transform_lang_string_name_type_with_context(*that:* LangStringNameType, *context: ContextT*) → T

Transform `that` in `context`.

transform_lang_string_text_type_with_context(*that:* LangStringTextType, *context: ContextT*) → T

Transform `that` in `context`.

transform_environment_with_context(*that:* Environment, *context: ContextT*) → T

Transform `that` in `context`.

transform_embedded_data_specification_with_context(*that:* EmbeddedDataSpecification, *context: ContextT*) → T

Transform `that` in `context`.

transform_level_type_with_context(*that:* LevelType, *context: ContextT*) → T

Transform `that` in `context`.

transform_value_reference_pair_with_context(*that:* ValueReferencePair, *context: ContextT*) → T

Transform `that` in `context`.

transform_value_list_with_context(*that:* ValueList, *context: ContextT*) → T

Transform `that` in `context`.

transform_lang_string_preferred_name_type_iec_61360_with_context(*that:* LangStringPreferredNameTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

transform_lang_string_short_name_type_iec_61360_with_context(*that:* LangStringShortNameTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

transform_lang_string_definition_type_iec_61360_with_context(*that:* LangStringDefinitionTypeIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

transform_data_specification_iec_61360_with_context(*that:* DataSpecificationIEC61360, *context: ContextT*) → T

Transform `that` in `context`.

1.3.6 aas_core3.verification

Verify that the instances of the meta-model satisfy the invariants.

Here is an example how to verify an instance of *aas_core3.types.Extension*:

```
import aas_core3.types as aas_types
import aas_core3.verification as aas_verification

an_instance = aas_types.Extension(
    # ... some constructor arguments ...
)

for error in aas_verification.verify(an_instance):
    print(f"{error.cause} at: {error.path}")
```

class aas_core3.verification.**PropertySegment**(*instance:* Class, *name: str*)

Represent a property access on a path to an erroneous value.

__init__(*instance:* Class, *name: str*) → None

Initialize with the given values.

instance: Final[*Class*]

Instance containing the property

name: Final[str]

Name of the property

__str__() → str

Return str(self).

class aas_core3.verification.**IndexSegment**(*sequence: Sequence[*Class*]*, *index: int*)

Represent an index access on a path to an erroneous value.

__init__(*sequence: Sequence[*Class*]*, *index: int*) → None

Initialize with the given values.

sequence: Final[Sequence[*Class*]]

Sequence containing the item at *index*

index: Final[int]

Index of the item

__str__() → str

Return str(self).

class aas_core3.verification.**Path**

Represent the relative path to the erroneous value.

__init__() → None

Initialize as an empty path.

property segments: Sequence[Union[*PropertySegment*, *IndexSegment*]]

Get the segments of the path.

__str__() → str

Return str(self).

class aas_core3.verification.**Error**(*cause: str*)

Represent a verification error in the data.

__init__(*cause: str*) → None

Initialize as an error with an empty path.

cause: Final[str]

Human-readable description of the error

path: Final[*Path*]

Path to the erroneous value

__repr__() → str

Return repr(self).

aas_core3.verification.**matches_id_short**(*text: str*) → bool

Check that `text` is a valid short ID.

aas_core3.verification.**matches_version_type**(*text: str*) → bool

Check that `text` is a valid version string.

aas_core3.verification.**matches_revision_type**(*text: str*) → bool

Check that `text` is a valid revision string.

aas_core3.verification.**matches_xs_date_time_utc**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:dateTime`.

The time zone must be fixed to UTC. We verify only that the `text` matches a pre-defined pattern. We *do not* verify that the day of month is correct nor do we check for leap seconds.

See: https://www.w3.org/TR/xmlschema-2/#dateTime

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**is_xs_date_time_utc**(*value: str*) → bool

Check that `value` is a `xs:dateTime` with the time zone set to UTC.

aas_core3.verification.**matches_mime_type**(*text: str*) → bool

Check that `text` conforms to the pattern of MIME type.

The definition has been taken from: https://www.rfc-editor.org/rfc/rfc7231#section-3.1.1.1, https://www.rfc-editor.org/rfc/rfc7230#section-3.2.3 and https://www.rfc-editor.org/rfc/rfc7230#section-3.2.6.

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_rfc_8089_path**(*text: str*) → bool

Check that `text` is a path conforming to the pattern of RFC 8089.

The definition has been taken from: https://datatracker.ietf.org/doc/html/rfc8089

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_bcp_47`**(*text: str*) → bool

Check that `text` is a valid BCP 47 language tag.

See: https://en.wikipedia.org/wiki/IETF_language_tag

`aas_core3.verification.`**`lang_strings_have_unique_languages`**(*lang_strings: Iterable[AbstractLangString]*) → bool

Check that `lang_strings` are specified each for a unique language.

`aas_core3.verification.`**`qualifier_types_are_unique`**(*qualifiers: Iterable[Qualifier]*) → bool

Check that there are no duplicate *`types.Qualifier.type`*'s in the `qualifiers`.

`aas_core3.verification.`**`matches_xml_serializable_string`**(*text: str*) → bool

Check that `text` conforms to the pattern of the Constraint AASd-130.

Ensures that encoding is possible and interoperability between different serializations is possible.

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_any_uri`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:anyURI`.

See: https://www.w3.org/TR/xmlschema-2/#anyURI and https://datatracker.ietf.org/doc/html/rfc3987

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_base_64_binary`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:base64Binary`.

See: https://www.w3.org/TR/xmlschema-2/#base64Binary

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_boolean`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:boolean`.

See: https://www.w3.org/TR/xmlschema-2/#boolean

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_date`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:date`.

See: https://www.w3.org/TR/xmlschema-2/#date

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_date_time`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:dateTime`.

See: https://www.w3.org/TR/xmlschema-2/#dateTime

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`is_xs_date_time`**(*value: str*) → bool

Check that `value` is a `xs:dateTime` with the time zone set to UTC.

`aas_core3.verification.`**`matches_xs_decimal`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:decimal`.

See: https://www.w3.org/TR/xmlschema-2/#decimal

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_double`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:double`.

See: https://www.w3.org/TR/xmlschema-2/#double

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_duration`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:duration`.

See: https://www.w3.org/TR/xmlschema-2/#duration

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_float`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:float`.

See: https://www.w3.org/TR/xmlschema-2/#float

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_g_day**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:gDay`.

See: https://www.w3.org/TR/xmlschema-2/#gDay

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_g_month**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:gMonth`.

See: https://www.w3.org/TR/xmlschema-2/#gMonth

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_g_month_day**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:gMonthDay`.

See: https://www.w3.org/TR/xmlschema-2/#gMonthDay

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_g_year**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:gYear`.

See: https://www.w3.org/TR/xmlschema-2/#gYear

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_g_year_month**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:gYearMonth`.

See: https://www.w3.org/TR/xmlschema-2/#gYearMonth

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_hex_binary**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:hexBinary`.

See: https://www.w3.org/TR/xmlschema-2/#hexBinary

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_time`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:time`.

See: https://www.w3.org/TR/xmlschema-2/#time

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_integer`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:integer`.

See: https://www.w3.org/TR/xmlschema-2/#integer

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_long`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:long`.

See: https://www.w3.org/TR/xmlschema-2/#long

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_int`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:int`.

See: https://www.w3.org/TR/xmlschema-2/#int

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_short`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:short`.

See: https://www.w3.org/TR/xmlschema-2/#short

Parameters
`text` – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**`matches_xs_byte`**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:byte`.

See: https://www.w3.org/TR/xmlschema-2/#byte

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**matches_xs_non_negative_integer**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:nonNegativeInteger`.

See: https://www.w3.org/TR/xmlschema-2/#nonNegativeInteger

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**matches_xs_positive_integer**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:positiveInteger`.

See: https://www.w3.org/TR/xmlschema-2/#positiveInteger

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**matches_xs_unsigned_long**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:unsignedLong`.

See: https://www.w3.org/TR/xmlschema-2/#unsignedLong

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**matches_xs_unsigned_int**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:unsignedInt`.

See: https://www.w3.org/TR/xmlschema-2/#unsignedInt

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

`aas_core3.verification.`**matches_xs_unsigned_short**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:unsignedShort`.

See: https://www.w3.org/TR/xmlschema-2/#unsignedShort

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_unsigned_byte**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:unsignedByte`.

See: https://www.w3.org/TR/xmlschema-2/#unsignedByte

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_non_positive_integer**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:nonPositiveInteger`.

See: https://www.w3.org/TR/xmlschema-2/#nonPositiveInteger

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_negative_integer**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:negativeInteger`.

See: https://www.w3.org/TR/xmlschema-2/#negativeInteger

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**matches_xs_string**(*text: str*) → bool

Check that `text` conforms to the pattern of an `xs:string`.

See: https://www.w3.org/TR/xmlschema-2/#string

Parameters
text – Text to be checked

Returns
True if the `text` conforms to the pattern

aas_core3.verification.**is_xs_date**(*value: str*) → bool

Check that `value` is a valid `xs:date`.

aas_core3.verification.**is_xs_double**(*value: str*) → bool

Check that `value` is a valid `xs:double`.

aas_core3.verification.**is_xs_float**(*value: str*) → bool

Check that `value` is a valid `xs:float`.

aas_core3.verification.**is_xs_g_month_day**(*value: str*) → bool

Check that `value` is a valid `xs:gMonthDay`.

aas_core3.verification.**is_xs_long**(*value: str*) → bool

Check that `value` is a valid `xs:long`.

aas_core3.verification.**is_xs_int**(*value: str*) → bool

Check that `value` is a valid `xs:int`.

aas_core3.verification.**is_xs_short**(*value: str*) → bool

Check that `value` is a valid `xs:short`.

aas_core3.verification.**is_xs_byte**(*value: str*) → bool

Check that `value` is a valid `xs:byte`.

aas_core3.verification.**is_xs_unsigned_long**(*value: str*) → bool

Check that `value` is a valid `xs:unsignedLong`.

aas_core3.verification.**is_xs_unsigned_int**(*value: str*) → bool

Check that `value` is a valid `xs:unsignedInt`.

aas_core3.verification.**is_xs_unsigned_short**(*value: str*) → bool

Check that `value` is a valid `xs:unsignedShort`.

aas_core3.verification.**is_xs_unsigned_byte**(*value: str*) → bool

Check that `value` is a valid `xs:unsignedByte`.

aas_core3.verification.**value_consistent_with_xsd_type**(*value: str*, *value_type:* DataTypeDefXSD) → bool

Check that `value` is consistent with the given `value_type`.

aas_core3.verification.**is_model_reference_to**(*reference:* Reference, *expected_type:* KeyTypes) → bool

Check that the target of the model reference matches the `expected_type`.

aas_core3.verification.**is_model_reference_to_referable**(*reference:* Reference) → bool

Check that the target of the reference matches a `constants.AAS_REFERABLES`.

aas_core3.verification.**id_shorts_are_unique**(*referables: Iterable[*Referable*]*) → bool

Check that all *types.Referable.id_short* are unique among `referables`.

aas_core3.verification.**id_shorts_of_variables_are_unique**(*input_variables: Optional[List[*OperationVariable*]]*, *output_variables: Optional[List[*OperationVariable*]]*, *inoutput_variables: Optional[List[*OperationVariable*]]*) → bool

Check that the *types.Referable.id_short*'s among all the `input_variables`, `output_variables` and `inoutput_variables` are unique.

aas_core3.verification.**extension_names_are_unique**(*extensions: Iterable[*Extension*]*) → bool

Check that all *types.Extension.name* are unique among `extensions`.

aas_core3.verification.**submodel_elements_have_identical_semantic_ids**(*elements: Iterable[*SubmodelElement*]*) → bool

Check that all `elements` have the identical *types.HasSemantics.semantic_id*.

aas_core3.verification.**submodel_element_is_of_type**(*element:* SubmodelElement, *expected_type:* AASSubmodelElements) → bool

Check that `element` is an instance of class corresponding to `expected_type`.

aas_core3.verification.**properties_or_ranges_have_value_type**(*elements: Iterable[*SubmodelElement*]*, *value_type:* DataTypeDefXSD) → bool

Check that `elements` which are *types.Property* or *types.Range* have the given `value_type`.

aas_core3.verification.**reference_key_values_equal**(*that:* Reference, *other:* Reference) → bool

Check that the two references, `that` and `other`, are equal by comparing their *types.Reference.keys* by *types.Key.value*'s.

aas_core3.verification.**data_specification_iec_61360s_for_property_or_value_have_appropriate_data_type**(*en*
It
er
ab
→
bo

Check that *types.DataSpecificationIEC61360.data_type* is defined appropriately for all data specifications whose content is given as IEC 61360.

aas_core3.verification.**data_specification_iec_61360s_for_reference_have_appropriate_data_type**(*embedded_d*
It-
er-
*able[*Embed
→
bool

Check that *types.DataSpecificationIEC61360.data_type* is defined appropriately for all data specifications whose content is given as IEC 61360.

aas_core3.verification.**data_specification_iec_61360s_for_document_have_appropriate_data_type**(*embedded_da*
It-
er-
*able[*Embedd
→
bool

Check that *types.DataSpecificationIEC61360.data_type* is defined appropriately for all data specifications whose content is given as IEC 61360.

aas_core3.verification.**data_specification_iec_61360s_have_data_type**(*embedded_data_specifications:*
Iter-
*able[*EmbeddedDataSpecification*]*)
→ bool

Check that *types.DataSpecificationIEC61360.data_type* is defined for all data specifications whose content is given as IEC 61360.

aas_core3.verification.**data_specification_iec_61360s_have_value**(*embedded_data_specifications:*
Iter-
*able[*EmbeddedDataSpecification*]*)
→ bool

Check that *types.DataSpecificationIEC61360.value* is defined for all data specifications whose content is given as IEC 61360.

aas_core3.verification.**data_specification_iec_61360s_have_definition_at_least_in_english**(*embedded_data_sp*
It-
er-
*able[*EmbeddedDa
→
bool

Check that *types.DataSpecificationIEC61360.definition* is defined for all data specifications whose content is given as IEC 61360 at least in English.

aas_core3.verification.**is_bcp_47_for_english**(*text: str*) → bool

Check that the `text` corresponds to a BCP47 code for english.

aas_core3.verification.**verify**(*that:* Class) → Iterator[*Error*]

Verify the constraints of `that` recursively.

Parameters

that¶ – instance whose constraints we want to verify

Yield

constraint violations

aas_core3.verification.**verify_non_empty_xml_serializable_string**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_date_time_utc**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_duration**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_blob_type**(*that: bytes*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_identifier**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_value_type_iec_61360**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_name_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_version_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_revision_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_label_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_message_topic_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_bcp_47_language_tag**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_content_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_path_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_qualifier_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_value_data_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

aas_core3.verification.**verify_id_short_type**(*that: str*) → Iterator[*Error*]

Verify the constraints of `that`.

1.3.7 aas_core3.xmlization

Read and write AAS models as XML.

For reading, we provide different reading functions, each handling a different kind of input. All the reading functions operate in one pass, *i.e.*, the source is read incrementally and the complete XML is not held in memory.

We provide the following four reading functions (where `X` represents the name of the class):

1) `X_from_iterparse` reads from a stream of `(event, element)` tuples coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`. If you do not trust the source, please consider using defusedxml.ElementTree.
2) `X_from_stream` reads from the given text stream.
3) `X_from_file` reads from a file on disk.
4) `X_from_str` reads from the given string.

The functions `X_from_stream`, `X_from_file` and `X_from_str` provide an extra parameter, `has_iterparse`, which allows you to use a parsing library different from `xml.etree.ElementTree`. For example, you can pass in defusedxml.ElementTree.

All XML elements are expected to live in the *NAMESPACE*.

For writing, use the function *aas_core3.xmlization.write()* which translates the instance of the model into an XML document and writes it in one pass to the stream.

Here is an example usage how to de-serialize from a file:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.read_extension_from_file(
    path
)

# Do something with the ``instance``
```

Here is another code example where we serialize the instance:

```
import pathlib

import aas_core3.types as aas_types
import aas_core3.xmlization as aas_xmlization

instance = Extension(
    ... # some constructor arguments
)

pth = pathlib.Path(...)
```

(continues on next page)

(continued from previous page)

```
with pth.open("wt") as fid:
    aas_xmlization.write(instance, fid)
```

aas_core3.xmlization.**NAMESPACE = 'https://admin-shell.io/aas/3/0'**

XML namespace in which all the elements are expected to reside

class aas_core3.xmlization.**Element**(**args*, ***kwargs*)

Behave like `xml.etree.ElementTree.Element()`.

property attrib: Optional[Mapping[str, str]]

Attributes of the element

property text: Optional[str]

Text content of the element

property tail: Optional[str]

Tail text of the element

property tag: str

Tag of the element; with a namespace provided as a `{...}` prefix

clear() → None

Behave like `xml.etree.ElementTree.Element.clear()`.

__init__(**args*, ***kwargs*)

class aas_core3.xmlization.**HasIterparse**(**args*, ***kwargs*)

Parse an XML document incrementally.

iterparse(*source: TextIO*, *events: Optional[Sequence[str]] = None*) → Iterator[Tuple[str, *Element*]]

Behave like `xml.etree.ElementTree.iterparse()`.

__init__(**args*, ***kwargs*)

class aas_core3.xmlization.**ElementSegment**(*element:* Element)

Represent an element on a path to the erroneous value.

__init__(*element:* Element) → None

Initialize with the given values.

element: Final[*Element*]

Erroneous element

__str__() → str

Render the segment as a tag without the namespace.

We deliberately omit the namespace in the tag names. If you want to actually query with the resulting XPath, you have to insert the namespaces manually. We did not know how to include the namespace in a meaningful way, as XPath assumes namespace prefixes to be defined *outside* of the document. At least the path thus rendered is informative, and you should be able to descend it manually.

class aas_core3.xmlization.**IndexSegment**(*element:* Element, *index: int*)

Represent an element in a sequence on a path to the erroneous value.

__init__(*element:* Element, *index: int*) → None

Initialize with the given values.

element: Final[*Element*]

Erroneous element

index: Final[int]

Index of the element in the sequence

__str__() → str

Render the segment as an element wildcard with the index.

class aas_core3.xmlization.**Path**

Represent the relative path to the erroneous element.

__init__() → None

Initialize as an empty path.

property segments: Sequence[Union[*ElementSegment*, *IndexSegment*]]

Get the segments of the path.

__str__() → str

Render the path as a relative XPath.

We omit the leading / so that you can easily prefix it as you need.

exception aas_core3.xmlization.**DeserializationException**(*cause: str*)

Signal that the XML de-serialization could not be performed.

__init__(*cause: str*) → None

Initialize with the given cause and an empty path.

cause: Final[str]

Human-readable explanation of the exception's cause

path: Final[*Path*]

Relative path to the erroneous value

aas_core3.xmlization.**has_semantics_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *HasSemantics*

Read an instance of *types.HasSemantics* from the iterator.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.has_semantics_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasSemantics* read from `iterator`

aas_core3.xmlization.**has_semantics_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementT*
→ *HasSemantics*

Read an instance of *types.HasSemantics* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.has_semantics_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters
- **stream**¶ – representing an instance of *types.HasSemantics* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasSemantics* read from `stream`

aas_core3.xmlization.**has_semantics_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*
→ *HasSemantics*

Read an instance of *types.HasSemantics* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.has_semantics_from_file(
```

(continues on next page)

(continued from previous page)

```
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.HasSemantics* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasSemantics* read from `path`

aas_core3.xmlization.**has_semantics_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.p*) → *HasSemantics*

Read an instance of *types.HasSemantics* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.has_semantics_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.HasSemantics* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasSemantics* read from `text`

aas_core3.xmlization.**extension_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *Extension*

Read an instance of *types.Extension* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.extension_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Extension* read from `iterator`

aas_core3.xmlization.**extension_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *Extension*

Read an instance of *types.Extension* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.extension_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Extension* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Extension* read from `stream`

aas_core3.xmlization.**extension_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Extension*

Read an instance of *types.Extension* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.extension_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Extension* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Extension* read from `path`

aas_core3.xmlization.**extension_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Extension*

Read an instance of *types.Extension* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.extension_from_str(
    text
```

(continues on next page)

(continued from previous page)

```
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Extension* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Extension* read from `text`

aas_core3.xmlization.**has_extensions_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *HasExtensions*

Read an instance of *types.HasExtensions* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.has_extensions_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasExtensions* read from `iterator`

aas_core3.xmlization.**has_extensions_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Element*
→ *HasExtensions*

Read an instance of *types.HasExtensions* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.has_extensions_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.HasExtensions* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasExtensions* read from `stream`

aas_core3.xmlization.**has_extensions_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTre*
→ *HasExtensions*

Read an instance of *types.HasExtensions* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.has_extensions_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.HasExtensions* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasExtensions* read from `path`

aas_core3.xmlization.**has_extensions_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*) → *HasExtensions*

Read an instance of *types.HasExtensions* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.has_extensions_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.HasExtensions* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasExtensions* read from `text`

aas_core3.xmlization.**referable_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Referable*

Read an instance of *types.Referable* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
```

(continues on next page)

(continued from previous page)

```
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.referable_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Referable* read from `iterator`

aas_core3.xmlization.**referable_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*) → *Referable*

Read an instance of *types.Referable* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.referable_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Referable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Referable* read from `stream`

aas_core3.xmlization.**referable_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Referable*

Read an instance of *types.Referable* from the path.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.referable_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Referable* in XML
- **has_iterparse**¶ – Module containing iterparse function.

  Default is to use xml.etree.ElementTree from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Referable* read from path

aas_core3.xmlization.**referable_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Referable*

Read an instance of *types.Referable* from the text.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.referable_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Referable* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Referable* read from `text`

aas_core3.xmlization.**identifiable_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Identifiable*

Read an instance of *types.Identifiable* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.identifiable_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Identifiable* read from `iterator`

aas_core3.xmlization.**identifiable_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTre*
→ *Identifiable*

Read an instance of *types.Identifiable* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
```

(continues on next page)

(continued from previous page)

```
    instance = aas_xmlization.identifiable_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Identifiable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Identifiable* read from `stream`

aas_core3.xmlization.**identifiable_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.p*) → *Identifiable*

Read an instance of *types.Identifiable* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.identifiable_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Identifiable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Identifiable* read from `path`

aas_core3.xmlization.**identifiable_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*) → *Identifiable*

Read an instance of *types.Identifiable* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.identifiable_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Identifiable* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Identifiable* read from `text`

aas_core3.xmlization.**has_kind_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *HasKind*

Read an instance of *types.HasKind* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.has_kind_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasKind* read from `iterator`

aas_core3.xmlization.**has_kind_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *HasKind*

Read an instance of *types.HasKind* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.has_kind_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters
- **stream**¶ – representing an instance of *types.HasKind* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasKind* read from `stream`

aas_core3.xmlization.**has_kind_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *HasKind*

Read an instance of *types.HasKind* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.has_kind_from_file(
```

(continues on next page)

(continued from previous page)

```
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.HasKind* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasKind* read from `path`

aas_core3.xmlization.**has_kind_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *HasKind*

Read an instance of *types.HasKind* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.has_kind_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.HasKind* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasKind* read from `text`

aas_core3.xmlization.**has_data_specification_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *HasDataSpecification*

Read an instance of *types.HasDataSpecification* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.has_data_specification_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasDataSpecification* read from `iterator`

aas_core3.xmlization.**has_data_specification_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etre*
→ *HasDataSpecification*

Read an instance of *types.HasDataSpecification* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.has_data_specification_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.HasDataSpecification* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasDataSpecification* read from `stream`

aas_core3.xmlization.**has_data_specification_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/*
→ *HasDataSpecification*

Read an instance of *types.HasDataSpecification* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.has_data_specification_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.HasDataSpecification* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.HasDataSpecification* read from `path`

aas_core3.xmlization.**has_data_specification_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *HasDataSpecification*

Read an instance of *types.HasDataSpecification* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.has_data_specification_from_str(
    text
)
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.HasDataSpecification* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.HasDataSpecification* read from `text`

aas_core3.xmlization.**administrative_information_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *AdministrativeInformation*

Read an instance of *types.AdministrativeInformation* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.administrative_information_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AdministrativeInformation* read from `iterator`

aas_core3.xmlization.**administrative_information_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xm*) → *AdministrativeInformation*

Read an instance of *types.AdministrativeInformation* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.administrative_information_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.AdministrativeInformation* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AdministrativeInformation* read from `stream`

aas_core3.xmlization.**administrative_information_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/e*) → *AdministrativeInformation*

Read an instance of *types.AdministrativeInformation* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.administrative_information_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.AdministrativeInformation* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AdministrativeInformation* read from `path`

aas_core3.xmlization.**administrative_information_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/et*
→ *AdministrativeInformation*

Read an instance of *types.AdministrativeInformation* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.administrative_information_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.AdministrativeInformation* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AdministrativeInformation* read from `text`

aas_core3.xmlization.**qualifiable_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Qualifiable*

Read an instance of *types.Qualifiable* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.qualifiable_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifiable* read from `iterator`

aas_core3.xmlization.**qualifiable_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*
→ *Qualifiable*

Read an instance of *types.Qualifiable* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.qualifiable_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Qualifiable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifiable* read from `stream`

aas_core3.xmlization.**qualifiable_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *Qualifiable*

Read an instance of *types.Qualifiable* from the path.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.qualifiable_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Qualifiable* in XML
- **has_iterparse**¶ – Module containing iterparse function.

  Default is to use xml.etree.ElementTree from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifiable* read from path

aas_core3.xmlization.**qualifiable_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*
→ *Qualifiable*

Read an instance of *types.Qualifiable* from the text.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.qualifiable_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Qualifiable* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Qualifiable* read from `text`

aas_core3.xmlization.**qualifier_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Qualifier*

Read an instance of *types.Qualifier* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.qualifier_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Qualifier* read from `iterator`

aas_core3.xmlization.**qualifier_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *Qualifier*

Read an instance of *types.Qualifier* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.qualifier_from_stream(
```

(continues on next page)

(continued from previous page)

```
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.Qualifier* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifier* read from `stream`

aas_core3.xmlization.**qualifier_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Qualifier*

Read an instance of *types.Qualifier* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.qualifier_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Qualifier* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifier* read from `path`

aas_core3.xmlization.**qualifier_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Qualifier*

Read an instance of *types.Qualifier* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.qualifier_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Qualifier* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Qualifier* read from `text`

aas_core3.xmlization.**asset_administration_shell_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *AssetAdministrationShell*

Read an instance of *types.AssetAdministrationShell* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.asset_administration_shell_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AssetAdministrationShell* read from `iterator`

aas_core3.xmlization.**asset_administration_shell_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xm*
→ *AssetAdministrationShell*

Read an instance of *types.AssetAdministrationShell* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.asset_administration_shell_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.AssetAdministrationShell* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AssetAdministrationShell* read from `stream`

aas_core3.xmlization.**asset_administration_shell_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/e*
→ *AssetAdministrationShell*

Read an instance of *types.AssetAdministrationShell* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.asset_administration_shell_from_file(
    path
```

(continues on next page)

(continued from previous page)

```
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.AssetAdministrationShell* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AssetAdministrationShell* read from `path`

aas_core3.xmlization.**asset_administration_shell_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/et*) → *AssetAdministrationShell*

Read an instance of *types.AssetAdministrationShell* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.asset_administration_shell_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.AssetAdministrationShell* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AssetAdministrationShell* read from `text`

aas_core3.xmlization.**asset_information_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *AssetInformation*

Read an instance of *types.AssetInformation* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.asset_information_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AssetInformation* read from `iterator`

aas_core3.xmlization.**asset_information_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elem*
→ *AssetInformation*

Read an instance of *types.AssetInformation* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.asset_information_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.AssetInformation* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AssetInformation* read from `stream`

aas_core3.xmlization.**asset_information_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Eleme*
→ *AssetInformation*

Read an instance of *types.AssetInformation* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.asset_information_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.AssetInformation* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AssetInformation* read from `path`

aas_core3.xmlization.**asset_information_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Element*
→ *AssetInformation*

Read an instance of *types.AssetInformation* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.asset_information_from_str(
```

(continues on next page)

(continued from previous page)

```
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.AssetInformation* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AssetInformation* read from `text`

aas_core3.xmlization.**resource_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Resource*

Read an instance of *types.Resource* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.resource_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Resource* read from `iterator`

aas_core3.xmlization.**resource_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *Resource*

Read an instance of *types.Resource* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.resource_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Resource* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Resource* read from `stream`

aas_core3.xmlization.**resource_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Resource*

Read an instance of *types.Resource* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.resource_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Resource* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Resource* read from `path`

aas_core3.xmlization.**resource_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Resource*

Read an instance of *types.Resource* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.resource_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Resource* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Resource* read from `text`

aas_core3.xmlization.**specific_asset_id_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *SpecificAssetID*

Read an instance of *types.SpecificAssetID* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
```

(continues on next page)

(continued from previous page)

```
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.specific_asset_id_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SpecificAssetID* read from `iterator`

aas_core3.xmlization.**specific_asset_id_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elem*
→ *SpecificAssetID*

Read an instance of *types.SpecificAssetID* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.specific_asset_id_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.SpecificAssetID* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SpecificAssetID* read from `stream`

aas_core3.xmlization.**specific_asset_id_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elemer*
→ *SpecificAssetID*

Read an instance of *types.SpecificAssetID* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.specific_asset_id_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.SpecificAssetID* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SpecificAssetID* read from `path`

aas_core3.xmlization.**specific_asset_id_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Element*
→ *SpecificAssetID*

Read an instance of *types.SpecificAssetID* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.specific_asset_id_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.SpecificAssetID* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SpecificAssetID* read from `text`

aas_core3.xmlization.**submodel_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Submodel*

Read an instance of *types.Submodel* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.submodel_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Submodel* read from `iterator`

aas_core3.xmlization.**submodel_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *Submodel*

Read an instance of *types.Submodel* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.submodel_from_stream(
```

(continues on next page)

(continued from previous page)

```
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.Submodel* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Submodel* read from `stream`

aas_core3.xmlization.**submodel_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Submodel*

Read an instance of *types.Submodel* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.submodel_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Submodel* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Submodel* read from `path`

aas_core3.xmlization.**submodel_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Submodel*

Read an instance of *types.Submodel* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.submodel_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Submodel* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Submodel* read from `text`

aas_core3.xmlization.**submodel_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *SubmodelElement*

Read an instance of *types.SubmodelElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.submodel_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElement* read from `iterator`

aas_core3.xmlization.**submodel_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Eleme*
→ *SubmodelElement*

Read an instance of *types.SubmodelElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.submodel_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.SubmodelElement* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElement* read from `stream`

aas_core3.xmlization.**submodel_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Element*
→ *SubmodelElement*

Read an instance of *types.SubmodelElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.submodel_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.SubmodelElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElement* read from `path`

aas_core3.xmlization.**submodel_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementT*) → *SubmodelElement*

Read an instance of *types.SubmodelElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.submodel_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.SubmodelElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElement* read from `text`

aas_core3.xmlization.**relationship_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *RelationshipElement*

Read an instance of *types.RelationshipElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET
```

(continues on next page)

(continued from previous page)

```
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.relationship_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.RelationshipElement* read from `iterator`

aas_core3.xmlization.**relationship_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/*
→ *RelationshipElement*

Read an instance of *types.RelationshipElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.relationship_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.RelationshipElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.RelationshipElement* read from `stream`

aas_core3.xmlization.**relationship_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*
→ *RelationshipElement*

Read an instance of *types.RelationshipElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.relationship_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.RelationshipElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.RelationshipElement* read from `path`

aas_core3.xmlization.**relationship_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elen*
→ *RelationshipElement*

Read an instance of *types.RelationshipElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.relationship_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.RelationshipElement* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.RelationshipElement* read from `text`

aas_core3.xmlization.**submodel_element_list_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *SubmodelElementList*

Read an instance of *types.SubmodelElementList* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.submodel_element_list_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElementList* read from `iterator`

aas_core3.xmlization.**submodel_element_list_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree*
→ *SubmodelElementList*

Read an instance of *types.SubmodelElementList* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
```

(continues on next page)

(continued from previous page)

```
    instance = aas_xmlization.submodel_element_list_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.SubmodelElementList* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElementList* read from `stream`

aas_core3.xmlization.**submodel_element_list_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *SubmodelElementList*

Read an instance of *types.SubmodelElementList* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.submodel_element_list_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.SubmodelElementList* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElementList* read from `path`

aas_core3.xmlization.**submodel_element_list_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*) → *SubmodelElementList*

Read an instance of *types.SubmodelElementList* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.submodel_element_list_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.SubmodelElementList* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElementList* read from `text`

aas_core3.xmlization.**submodel_element_collection_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *SubmodelElementCollection*

Read an instance of *types.SubmodelElementCollection* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.submodel_element_collection_from_iterparse(
        iterator
    )
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElementCollection* read from `iterator`

aas_core3.xmlization.**submodel_element_collection_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/x*
→ *SubmodelElementCollection*

Read an instance of *types.SubmodelElementCollection* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.submodel_element_collection_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.SubmodelElementCollection* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElementCollection* read from `stream`

aas_core3.xmlization.**submodel_element_collection_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml*
→ *SubmodelElementCollection*

Read an instance of *types.SubmodelElementCollection* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.submodel_element_collection_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.SubmodelElementCollection* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.SubmodelElementCollection* read from `path`

aas_core3.xmlization.**submodel_element_collection_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/e*) → *SubmodelElementCollection*

Read an instance of *types.SubmodelElementCollection* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.submodel_element_collection_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.SubmodelElementCollection* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns
Instance of *types.SubmodelElementCollection* read from `text`

aas_core3.xmlization.**data_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *DataElement*

Read an instance of *types.DataElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.data_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataElement* read from `iterator`

aas_core3.xmlization.**data_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTre*
→ *DataElement*

Read an instance of *types.DataElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.data_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.DataElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataElement* read from `stream`

aas_core3.xmlization.**data_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.p*
→ *DataElement*

Read an instance of *types.DataElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.data_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.DataElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataElement* read from `path`

aas_core3.xmlization.**data_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *DataElement*

Read an instance of *types.DataElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
text = "<...>...</...>"
instance = aas_xmlization.data_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.DataElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataElement* read from `text`

aas_core3.xmlization.**property_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Property*

Read an instance of *types.Property* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.property_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Property* read from `iterator`

aas_core3.xmlization.**property_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *Property*

Read an instance of *types.Property* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.property_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.Property* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Property* read from `stream`

aas_core3.xmlization.**property_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Property*

Read an instance of *types.Property* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.property_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Property* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Property* read from `path`

aas_core3.xmlization.**property_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Property*

Read an instance of *types.Property* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.property_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Property* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Property* read from `text`

aas_core3.xmlization.**multi_language_property_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *MultiLanguageProperty*

Read an instance of *types.MultiLanguageProperty* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
```

(continues on next page)

(continued from previous page)

```
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.multi_language_property_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.MultiLanguageProperty* read from `iterator`

aas_core3.xmlization.**multi_language_property_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/et*
→ *MultiLanguageProperty*

Read an instance of *types.MultiLanguageProperty* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.multi_language_property_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.MultiLanguageProperty* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.MultiLanguageProperty* read from `stream`

aas_core3.xmlization.**multi_language_property_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree*
→ *MultiLanguageProperty*

Read an instance of *types.MultiLanguageProperty* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.multi_language_property_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.MultiLanguageProperty* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.MultiLanguageProperty* read from `path`

aas_core3.xmlization.**multi_language_property_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/*
→ *MultiLanguageProperty*

Read an instance of *types.MultiLanguageProperty* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.multi_language_property_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.MultiLanguageProperty* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.MultiLanguageProperty* read from `text`

aas_core3.xmlization.**range_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Range*

Read an instance of *types.Range* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.range_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Range* read from `iterator`

aas_core3.xmlization.**range_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Range*

Read an instance of *types.Range* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.range_from_stream(
```

(continues on next page)

(continued from previous page)

```
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Range* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Range* read from `stream`

aas_core3.xmlization.**range_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Range*

Read an instance of *types.Range* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.range_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Range* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Range* read from `path`

aas_core3.xmlization.**range_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Range*

Read an instance of *types.Range* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.range_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Range* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Range* read from `text`

aas_core3.xmlization.**reference_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *ReferenceElement*

Read an instance of *types.ReferenceElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.reference_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ReferenceElement* read from `iterator`

aas_core3.xmlization.**reference_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elem*
→ *ReferenceElement*

Read an instance of *types.ReferenceElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.reference_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.ReferenceElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ReferenceElement* read from `stream`

aas_core3.xmlization.**reference_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elemer*
→ *ReferenceElement*

Read an instance of *types.ReferenceElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.reference_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.ReferenceElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ReferenceElement* read from `path`

aas_core3.xmlization.**reference_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Element*) → *ReferenceElement*

Read an instance of *types.ReferenceElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.reference_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.ReferenceElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ReferenceElement* read from `text`

aas_core3.xmlization.**blob_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Blob*

Read an instance of *types.Blob* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.blob_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Blob* read from `iterator`

aas_core3.xmlization.**blob_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Blob*

Read an instance of *types.Blob* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.blob_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Blob* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Blob* read from `stream`

aas_core3.xmlization.**blob_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Blob*

Read an instance of *types.Blob* from the path.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.blob_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Blob* in XML
- **has_iterparse** – Module containing iterparse function.

  Default is to use xml.etree.ElementTree from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Blob* read from path

aas_core3.xmlization.**blob_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Blob*

Read an instance of *types.Blob* from the text.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.blob_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Blob* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Blob* read from `text`

aas_core3.xmlization.**file_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *File*

Read an instance of *types.File* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.file_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.File* read from `iterator`

aas_core3.xmlization.**file_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *File*

Read an instance of *types.File* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.file_from_stream(
```

(continues on next page)

(continued from previous page)

```
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.File* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.File* read from `stream`

aas_core3.xmlization.**file_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *File*

Read an instance of *types.File* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.file_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.File* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.File* read from `path`

aas_core3.xmlization.**file_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *File*

Read an instance of *types.File* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.file_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.File* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.File* read from `text`

aas_core3.xmlization.**annotated_relationship_element_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *AnnotatedRelationshipElement*

Read an instance of *types.AnnotatedRelationshipElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.annotated_relationship_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AnnotatedRelationshipElement* read from `iterator`

aas_core3.xmlization.**annotated_relationship_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3* → *AnnotatedRelationshipElement*

Read an instance of *types.AnnotatedRelationshipElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.annotated_relationship_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.AnnotatedRelationshipElement* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AnnotatedRelationshipElement* read from `stream`

aas_core3.xmlization.**annotated_relationship_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/* → *AnnotatedRelationshipElement*

Read an instance of *types.AnnotatedRelationshipElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
path = pathlib.Path(...)
instance = aas_xmlization.annotated_relationship_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.AnnotatedRelationshipElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AnnotatedRelationshipElement* read from `path`

aas_core3.xmlization.**annotated_relationship_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/x*
→ *AnnotatedRelationshipElement*

Read an instance of *types.AnnotatedRelationshipElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.annotated_relationship_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.AnnotatedRelationshipElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AnnotatedRelationshipElement* read from `text`

aas_core3.xmlization.**entity_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *Entity*

Read an instance of *types.Entity* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.entity_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Entity* read from `iterator`

aas_core3.xmlization.**entity_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Entity*

Read an instance of *types.Entity* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.entity_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Entity* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Entity* read from `stream`

aas_core3.xmlization.**entity_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Entity*

Read an instance of *types.Entity* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.entity_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Entity* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Entity* read from `path`

aas_core3.xmlization.**entity_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Entity*

Read an instance of *types.Entity* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.entity_from_str(
    text
```

(continues on next page)

(continued from previous page)

```
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Entity* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Entity* read from `text`

aas_core3.xmlization.**event_payload_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *EventPayload*

Read an instance of *types.EventPayload* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.event_payload_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.EventPayload* read from `iterator`

aas_core3.xmlization.**event_payload_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementT*
→ *EventPayload*

Read an instance of *types.EventPayload* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.event_payload_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.EventPayload* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.EventPayload* read from `stream`

aas_core3.xmlization.**event_payload_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*
→ *EventPayload*

Read an instance of *types.EventPayload* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.event_payload_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.EventPayload* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EventPayload* read from `path`

aas_core3.xmlization.**event_payload_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.p*) → *EventPayload*

Read an instance of *types.EventPayload* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.event_payload_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.EventPayload* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EventPayload* read from `text`

aas_core3.xmlization.**event_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *EventElement*

Read an instance of *types.EventElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
```

(continues on next page)

(continued from previous page)

```
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.event_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EventElement* read from `iterator`

aas_core3.xmlization.**event_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementT*
→ *EventElement*

Read an instance of *types.EventElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.event_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.EventElement* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EventElement* read from `stream`

aas_core3.xmlization.**event_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*
→ *EventElement*

Read an instance of *types.EventElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.event_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.EventElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.EventElement* read from `path`

aas_core3.xmlization.**event_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.*
→ *EventElement*

Read an instance of *types.EventElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.event_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.EventElement* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EventElement* read from `text`

aas_core3.xmlization.**basic_event_element_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *BasicEventElement*

Read an instance of *types.BasicEventElement* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.basic_event_element_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.BasicEventElement* read from `iterator`

aas_core3.xmlization.**basic_event_element_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *BasicEventElement*

Read an instance of *types.BasicEventElement* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
```

(continues on next page)

(continued from previous page)

```
    instance = aas_xmlization.basic_event_element_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.BasicEventElement* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.BasicEventElement* read from `stream`

aas_core3.xmlization.**basic_event_element_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elen*
→ *BasicEventElement*

Read an instance of *types.BasicEventElement* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.basic_event_element_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.BasicEventElement* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.BasicEventElement* read from `path`

aas_core3.xmlization.**basic_event_element_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Eleme*) → *BasicEventElement*

Read an instance of *types.BasicEventElement* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.basic_event_element_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.BasicEventElement* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.BasicEventElement* read from `text`

aas_core3.xmlization.**operation_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Operation*

Read an instance of *types.Operation* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.operation_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Operation* read from `iterator`

aas_core3.xmlization.**operation_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *Operation*

Read an instance of *types.Operation* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.operation_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters
- **stream**¶ – representing an instance of *types.Operation* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Operation* read from `stream`

aas_core3.xmlization.**operation_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*
→ *Operation*

Read an instance of *types.Operation* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.operation_from_file(
```

(continues on next page)

(continued from previous page)

```
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Operation* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Operation* read from `path`

aas_core3.xmlization.**operation_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Operation*

Read an instance of *types.Operation* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.operation_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Operation* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Operation* read from `text`

aas_core3.xmlization.**operation_variable_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *OperationVariable*

Read an instance of *types.OperationVariable* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.operation_variable_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.OperationVariable* read from `iterator`

aas_core3.xmlization.**operation_variable_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*) → *OperationVariable*

Read an instance of *types.OperationVariable* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.operation_variable_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.OperationVariable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.OperationVariable* read from `stream`

aas_core3.xmlization.**operation_variable_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Eleme*) → *OperationVariable*

Read an instance of *types.OperationVariable* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.operation_variable_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.OperationVariable* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.OperationVariable* read from `path`

aas_core3.xmlization.**operation_variable_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elemer*) → *OperationVariable*

Read an instance of *types.OperationVariable* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.operation_variable_from_str(
    text
)
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.OperationVariable* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.OperationVariable* read from `text`

aas_core3.xmlization.**capability_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *Capability*

Read an instance of *types.Capability* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.capability_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Capability* read from `iterator`

aas_core3.xmlization.**capability_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.*
→ *Capability*

Read an instance of *types.Capability* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.capability_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Capability* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Capability* read from `stream`

aas_core3.xmlization.**capability_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *Capability*

Read an instance of *types.Capability* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.capability_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Capability* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns
Instance of *types.Capability* read from `path`

aas_core3.xmlization.**capability_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Capability*

Read an instance of *types.Capability* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.capability_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Capability* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Capability* read from `text`

aas_core3.xmlization.**concept_description_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *ConceptDescription*

Read an instance of *types.ConceptDescription* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.concept_description_from_iterparse(
        iterator
    )
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ConceptDescription* read from `iterator`

aas_core3.xmlization.**concept_description_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *ConceptDescription*

Read an instance of *types.ConceptDescription* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.concept_description_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.ConceptDescription* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ConceptDescription* read from `stream`

aas_core3.xmlization.**concept_description_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elen*
→ *ConceptDescription*

Read an instance of *types.ConceptDescription* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.concept_description_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.ConceptDescription* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ConceptDescription* read from `path`

aas_core3.xmlization.**concept_description_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Eleme*
→ *ConceptDescription*

Read an instance of *types.ConceptDescription* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.concept_description_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.ConceptDescription* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ConceptDescription* read from `text`

aas_core3.xmlization.**reference_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *Reference*

Read an instance of *types.Reference* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.reference_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Reference* read from `iterator`

aas_core3.xmlization.**reference_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*) → *Reference*

Read an instance of *types.Reference* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.reference_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Reference* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Reference* read from `stream`

aas_core3.xmlization.**reference_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Reference*

Read an instance of *types.Reference* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.reference_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Reference* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Reference* read from `path`

aas_core3.xmlization.**reference_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Reference*

Read an instance of *types.Reference* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.reference_from_str(
    text
```

(continues on next page)

(continued from previous page)

```
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Reference* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Reference* read from `text`

aas_core3.xmlization.**key_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Key*

Read an instance of *types.Key* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.key_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Key* read from `iterator`

aas_core3.xmlization.**key_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Key*

Read an instance of *types.Key* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.key_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.Key* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Key* read from `stream`

aas_core3.xmlization.**key_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Key*

Read an instance of *types.Key* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.key_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.Key* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns
Instance of *types.Key* read from `path`

aas_core3.xmlization.**key_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *Key*

Read an instance of *types.Key* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.key_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.Key* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Key* read from `text`

aas_core3.xmlization.**abstract_lang_string_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *AbstractLangString*

Read an instance of *types.AbstractLangString* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.abstract_lang_string_from_iterparse(
        iterator
    )
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AbstractLangString* read from `iterator`

aas_core3.xmlization.**abstract_lang_string_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/*
→ *AbstractLangString*

Read an instance of *types.AbstractLangString* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.abstract_lang_string_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.AbstractLangString* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.AbstractLangString* read from `stream`

aas_core3.xmlization.**abstract_lang_string_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*
→ *AbstractLangString*

Read an instance of *types.AbstractLangString* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.abstract_lang_string_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.AbstractLangString* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AbstractLangString* read from `path`

aas_core3.xmlization.**abstract_lang_string_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elem*
→ *AbstractLangString*

Read an instance of *types.AbstractLangString* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.abstract_lang_string_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.AbstractLangString* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.AbstractLangString* read from `text`

aas_core3.xmlization.**lang_string_name_type_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *LangStringNameType*

Read an instance of *types.LangStringNameType* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.lang_string_name_type_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringNameType* read from `iterator`

aas_core3.xmlization.**lang_string_name_type_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree*
→ *LangStringNameType*

Read an instance of *types.LangStringNameType* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.lang_string_name_type_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.LangStringNameType* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringNameType* read from `stream`

aas_core3.xmlization.**lang_string_name_type_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *LangStringNameType*

Read an instance of *types.LangStringNameType* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.lang_string_name_type_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.LangStringNameType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringNameType* read from `path`

aas_core3.xmlization.**lang_string_name_type_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*
→ *LangStringNameType*

Read an instance of *types.LangStringNameType* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
text = "<...>...</...>"
instance = aas_xmlization.lang_string_name_type_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.LangStringNameType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringNameType* read from `text`

aas_core3.xmlization.**lang_string_text_type_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *LangStringTextType*

Read an instance of *types.LangStringTextType* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.lang_string_text_type_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringTextType* read from `iterator`

aas_core3.xmlization.**lang_string_text_type_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree*
→ *LangStringTextType*

Read an instance of *types.LangStringTextType* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.lang_string_text_type_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.LangStringTextType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringTextType* read from `stream`

aas_core3.xmlization.**lang_string_text_type_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/E*
→ *LangStringTextType*

Read an instance of *types.LangStringTextType* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.lang_string_text_type_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.LangStringTextType* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringTextType* read from `path`

aas_core3.xmlization.**lang_string_text_type_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*) → *LangStringTextType*

Read an instance of *types.LangStringTextType* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.lang_string_text_type_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.LangStringTextType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringTextType* read from `text`

aas_core3.xmlization.**environment_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *Environment*

Read an instance of *types.Environment* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
```

(continues on next page)

(continued from previous page)

```
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.environment_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Environment* read from `iterator`

aas_core3.xmlization.**environment_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree*
→ *Environment*

Read an instance of *types.Environment* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.environment_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.Environment* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Environment* read from `stream`

aas_core3.xmlization.**environment_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py*
→ *Environment*

Read an instance of *types.Environment* from the path.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.environment_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.Environment* in XML
- **has_iterparse**¶ – Module containing iterparse function.

  Default is to use xml.etree.ElementTree from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.Environment* read from path

aas_core3.xmlization.**environment_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*
→ *Environment*

Read an instance of *types.Environment* from the text.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.environment_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.Environment* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.Environment* read from `text`

aas_core3.xmlization.**data_specification_content_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *DataSpecificationContent*

Read an instance of *types.DataSpecificationContent* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.data_specification_content_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataSpecificationContent* read from `iterator`

aas_core3.xmlization.**data_specification_content_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xm*) → *DataSpecificationContent*

Read an instance of *types.DataSpecificationContent* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.data_specification_content_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.DataSpecificationContent* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataSpecificationContent* read from `stream`

aas_core3.xmlization.**data_specification_content_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/e*) → *DataSpecificationContent*

Read an instance of *types.DataSpecificationContent* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.data_specification_content_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.DataSpecificationContent* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataSpecificationContent* read from `path`

aas_core3.xmlization.**data_specification_content_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/et*
→ *DataSpecificationContent*

Read an instance of *types.DataSpecificationContent* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.data_specification_content_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.DataSpecificationContent* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataSpecificationContent* read from `text`

aas_core3.xmlization.**embedded_data_specification_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *EmbeddedDataSpecification*

Read an instance of *types.EmbeddedDataSpecification* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.embedded_data_specification_from_iterparse(
        iterator
    )
```

(continues on next page)

(continued from previous page)

```
# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EmbeddedDataSpecification* read from `iterator`

aas_core3.xmlization.**embedded_data_specification_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/x*
→ *EmbeddedDataSpecification*

Read an instance of *types.EmbeddedDataSpecification* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.embedded_data_specification_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.EmbeddedDataSpecification* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.EmbeddedDataSpecification* read from `stream`

aas_core3.xmlization.**embedded_data_specification_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml*
→ *EmbeddedDataSpecification*

Read an instance of *types.EmbeddedDataSpecification* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.embedded_data_specification_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.EmbeddedDataSpecification* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.EmbeddedDataSpecification* read from `path`

aas_core3.xmlization.**embedded_data_specification_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/e*) → *EmbeddedDataSpecification*

Read an instance of *types.EmbeddedDataSpecification* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.embedded_data_specification_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.EmbeddedDataSpecification* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns
Instance of *types.EmbeddedDataSpecification* read from `text`

aas_core3.xmlization.**level_type_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *LevelType*

Read an instance of *types.LevelType* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.level_type_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LevelType* read from `iterator`

aas_core3.xmlization.**level_type_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.*
→ *LevelType*

Read an instance of *types.LevelType* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.level_type_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.LevelType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LevelType* read from `stream`

aas_core3.xmlization.**level_type_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *LevelType*

Read an instance of *types.LevelType* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.level_type_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.LevelType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LevelType* read from `path`

aas_core3.xmlization.**level_type_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *LevelType*

Read an instance of *types.LevelType* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
text = "<...>...</...>"
instance = aas_xmlization.level_type_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.LevelType* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LevelType* read from `text`

aas_core3.xmlization.**value_reference_pair_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *ValueReferencePair*

Read an instance of *types.ValueReferencePair* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.value_reference_pair_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ValueReferencePair* read from `iterator`

aas_core3.xmlization.**value_reference_pair_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/*
→ *ValueReferencePair*

Read an instance of *types.ValueReferencePair* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.value_reference_pair_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.ValueReferencePair* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ValueReferencePair* read from `stream`

aas_core3.xmlization.**value_reference_pair_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Ele*
→ *ValueReferencePair*

Read an instance of *types.ValueReferencePair* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.value_reference_pair_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.ValueReferencePair* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ValueReferencePair* read from `path`

aas_core3.xmlization.**value_reference_pair_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/Elem*) → *ValueReferencePair*

Read an instance of *types.ValueReferencePair* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.value_reference_pair_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.ValueReferencePair* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ValueReferencePair* read from `text`

aas_core3.xmlization.**value_list_from_iterparse**(*iterator: Iterator[Tuple[str,* Element*]]*) → *ValueList*

Read an instance of *types.ValueList* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
```

(continues on next page)

(continued from previous page)

```
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.value_list_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ValueList* read from `iterator`

aas_core3.xmlization.**value_list_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.*
→ *ValueList*

Read an instance of *types.ValueList* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.value_list_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.ValueList* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ValueList* read from `stream`

aas_core3.xmlization.**value_list_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'*) → *ValueList*

Read an instance of *types.ValueList* from the path.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.value_list_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.ValueList* in XML
- **has_iterparse**¶ – Module containing iterparse function.

  Default is to use xml.etree.ElementTree from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.ValueList* read from path

aas_core3.xmlization.**value_list_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml/etree/ElementTree.py'>*) → *ValueList*

Read an instance of *types.ValueList* from the text.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.value_list_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.ValueList* in XML

- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.ValueList* read from `text`

aas_core3.xmlization.**lang_string_preferred_name_type_iec_61360_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *LangStringPreferredNameTypeIEC61360*

Read an instance of *types.LangStringPreferredNameTypeIEC61360* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.lang_string_preferred_name_type_iec_61360_from_
↪iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringPreferredNameTypeIEC61360* read from `iterator`

aas_core3.xmlization.**lang_string_preferred_name_type_iec_61360_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.*
→ *LangStringPreferredNameTypeIEC61360*

Read an instance of *types.LangStringPreferredNameTypeIEC61360* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.lang_string_preferred_name_type_iec_61360_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.LangStringPreferredNameTypeIEC61360* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringPreferredNameTypeIEC61360* read from `stream`

aas_core3.xmlization.**lang_string_preferred_name_type_iec_61360_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18*
→ *LangStringPreferredNameTypeIEC61360*

Read an instance of *types.LangStringPreferredNameTypeIEC61360* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization
```

(continues on next page)

(continued from previous page)

```
path = pathlib.Path(...)
instance = aas_xmlization.lang_string_preferred_name_type_iec_61360_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.LangStringPreferredNameTypeIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringPreferredNameTypeIEC61360* read from `path`

aas_core3.xmlization.**lang_string_preferred_name_type_iec_61360_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/l* → *LangStringPreferredNameTypeIEC61360*

Read an instance of *types.LangStringPreferredNameTypeIEC61360* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.lang_string_preferred_name_type_iec_61360_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.LangStringPreferredNameTypeIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringPreferredNameTypeIEC61360* read from `text`

aas_core3.xmlization.**lang_string_short_name_type_iec_61360_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *LangStringShortNameTypeIEC61360*

Read an instance of *types.LangStringShortNameTypeIEC61360* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.lang_string_short_name_type_iec_61360_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters
iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringShortNameTypeIEC61360* read from `iterator`

aas_core3.xmlization.**lang_string_short_name_type_iec_61360_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib*
→ *LangStringShortNameTypeIEC61360*

Read an instance of *types.LangStringShortNameTypeIEC61360* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.lang_string_short_name_type_iec_61360_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.LangStringShortNameTypeIEC61360* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringShortNameTypeIEC61360* read from `stream`

aas_core3.xmlization.**lang_string_short_name_type_iec_61360_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/p*
→ *LangStringShortNameTypeIEC61360*

Read an instance of *types.LangStringShortNameTypeIEC61360* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.lang_string_short_name_type_iec_61360_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path** – to the file representing an instance of *types.LangStringShortNameTypeIEC61360* in XML
- **has_iterparse** – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringShortNameTypeIEC61360* read from `path`

aas_core3.xmlization.**lang_string_short_name_type_iec_61360_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/py*) → *LangStringShortNameTypeIEC61360*

Read an instance of *types.LangStringShortNameTypeIEC61360* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.lang_string_short_name_type_iec_61360_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.LangStringShortNameTypeIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringShortNameTypeIEC61360* read from `text`

aas_core3.xmlization.**lang_string_definition_type_iec_61360_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *LangStringDefinitionTypeIEC61360*

Read an instance of *types.LangStringDefinitionTypeIEC61360* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.lang_string_definition_type_iec_61360_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator¶ – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringDefinitionTypeIEC61360* read from `iterator`

aas_core3.xmlization.**lang_string_definition_type_iec_61360_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib*) → *LangStringDefinitionTypeIEC61360*

Read an instance of *types.LangStringDefinitionTypeIEC61360* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.lang_string_definition_type_iec_61360_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream**¶ – representing an instance of *types.LangStringDefinitionTypeIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringDefinitionTypeIEC61360* read from `stream`

aas_core3.xmlization.**lang_string_definition_type_iec_61360_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/p*
→ *LangStringDefinitionTypeIEC61360*

Read an instance of *types.LangStringDefinitionTypeIEC61360* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.lang_string_definition_type_iec_61360_from_file(
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.LangStringDefinitionTypeIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.LangStringDefinitionTypeIEC61360* read from `path`

aas_core3.xmlization.**lang_string_definition_type_iec_61360_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/py*
→ *LangStringDefinitionTypeIEC61360*

Read an instance of *types.LangStringDefinitionTypeIEC61360* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.lang_string_definition_type_iec_61360_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text** – representing an instance of *types.LangStringDefinitionTypeIEC61360* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.LangStringDefinitionTypeIEC61360* read from `text`

aas_core3.xmlization.**data_specification_iec_61360_from_iterparse**(*iterator: Iterator[Tuple[str, Element]]*) → *DataSpecificationIEC61360*

Read an instance of *types.DataSpecificationIEC61360* from the `iterator`.

Example usage:

```
import pathlib
import xml.etree.ElementTree as ET

import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
with path.open("rt") as fid:
    iterator = ET.iterparse(
        source=fid,
        events=['start', 'end']
    )
    instance = aas_xmlization.data_specification_iec_61360_from_iterparse(
        iterator
    )

# Do something with the ``instance``
```

Parameters

iterator – Input stream of `(event, element)` coming from `xml.etree.ElementTree.iterparse()` with the argument `events=["start", "end"]`

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataSpecificationIEC61360* read from `iterator`

aas_core3.xmlization.**data_specification_iec_61360_from_stream**(*stream: ~typing.TextIO*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/*) → *DataSpecificationIEC61360*

Read an instance of *types.DataSpecificationIEC61360* from the `stream`.

Example usage:

```
import aas_core3.xmlization as aas_xmlization

with open_some_stream_over_network(...) as stream:
    instance = aas_xmlization.data_specification_iec_61360_from_stream(
        stream
    )

# Do something with the ``instance``
```

Parameters

- **stream** – representing an instance of *types.DataSpecificationIEC61360* in XML
- **has_iterparse** – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise
DeserializationException if unexpected input

Returns
Instance of *types.DataSpecificationIEC61360* read from `stream`

aas_core3.xmlization.**data_specification_iec_61360_from_file**(*path: ~os.PathLike*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xm*) → *DataSpecificationIEC61360*

Read an instance of *types.DataSpecificationIEC61360* from the `path`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

path = pathlib.Path(...)
instance = aas_xmlization.data_specification_iec_61360_from_file(
```

(continues on next page)

(continued from previous page)

```
    path
)

# Do something with the ``instance``
```

Parameters

- **path**¶ – to the file representing an instance of *types.DataSpecificationIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataSpecificationIEC61360* read from `path`

aas_core3.xmlization.**data_specification_iec_61360_from_str**(*text: str*, *has_iterparse: ~aas_core3.xmlization.HasIterparse = <module 'xml.etree.ElementTree' from '/home/docs/.asdf/installs/python/3.8.18/lib/python3.8/xml*
→ *DataSpecificationIEC61360*

Read an instance of *types.DataSpecificationIEC61360* from the `text`.

Example usage:

```
import pathlib
import aas_core3.xmlization as aas_xmlization

text = "<...>...</...>"
instance = aas_xmlization.data_specification_iec_61360_from_str(
    text
)

# Do something with the ``instance``
```

Parameters

- **text**¶ – representing an instance of *types.DataSpecificationIEC61360* in XML
- **has_iterparse**¶ – Module containing `iterparse` function.

  Default is to use `xml.etree.ElementTree` from the standard library. If you have to deal with malicious input, consider using a library such as defusedxml.ElementTree.

Raise

DeserializationException if unexpected input

Returns

Instance of *types.DataSpecificationIEC61360* read from `text`

aas_core3.xmlization.**write**(*instance:* Class, *stream: TextIO*) → None

Write the XML representation of `instance` to `stream`.

Example usage:

```
import pathlib

import aas_core3.types as aas_types
import aas_core3.xmlization as aas_xmlization

instance = Extension(
    ... # some constructor arguments
)

pth = pathlib.Path(...)
with pth.open("wt") as fid:
    aas_xmlization.write(instance, fid)
```

Parameters

- **instance**¶ – to be serialized
- **stream**¶ – to write to

aas_core3.xmlization.**to_str**(*that:* Class) → str

Serialize `that` to an XML-encoded text.

Parameters

that¶ – instance to be serialized

Returns

`that` serialized to XML serialized to text

1.4 Design Decisions

We explain a couple of design decisions and trade-offs we deliberately made during the development of the SDK. These are our opinions — you may or may not agree, which is totally OK as there are always more than one way to do things and do them well.

However, the decisions elaborated here are not meant to convince you. We want to give you insight about why we did certain things, and why we didn't implement them in some other way.

1.4.1 Aggregations as Lists instead of Dictionaries

We decided to implement all the aggregations in the meta-model as list instead of dict.

Some structures just "scream" for a dictionary, such as submodel elements property in a *aas_core3.types.Submodel*. The submodel elements need to be unique w.r.t. their ID-shorts. So why didn't we model them as dictionaries, where keys are ID-shorts?

There are multiple reasons:

- "There are only two hard things in Computer Science: cache invalidation and naming things" (see this StackExchange). For example, the key in the dictionary and the `id_short` property of the submodel element need to be always in sync. Keeping such things in sync can be hard.

- When de-serializing, you need to hash on all the key/value pairs. In many situations, you do not perform any look-ups, but want to read the whole environment only once, and act upon it. Hashing would have wasted computational resources.
- You may want to index on more things than `id_short`. For example, retrieving submodel elements by their `semantic_id` is almost equally important.
- The order of the key/value pairs in a dictionary might not follow the order in the underlying serialized file. For example, if `dict` is used, the order is random. This would make the round-trip de-serialization serialization non-deterministic.
- Generating code based on dictionaries would have incurred additional complexity in aas-core-meta and aas-core-codegen as we would need to capture indexing in our machine-readable meta-models.

We therefore leave indexing (and syncing of the indices) to the user instead of pre-maturely providing a basic index on one of the features.

1.4.2 No Parent Child Associations

We did not model the parent child relations between the model elements for similar reasons why we did not implement dictionaries. Namely, keeping the associations in sync is hard. While you might have clear parent child relationship when you deserialize an environment, this relationship becomes less clear when you start re-using objects between environments.

Moreover, you need to sync the parent when an instance associated as its child is deleted. The complexity of this sync becomes hard (and computationally costly) as your object tree grows. What if you re-assign the instance to multiple parents?

For example, an instance of *aas_core3.types.Submodel* may appear in multiple instances of *aas_core3.types.Environment*. Which environment is the parent?

Multiple solutions are possible, and they depend on the application domain. In some cases, where you deal with static data, a simple dictionary parent child is sufficient. In other cases, more involved data structures and updating strategies are needed.

As we did not want to prejudice the SDK for a particular application domain, we left out parent child associations.

We indeed discussed a couple of concrete solutions, but failed to find a unifying approach which would satisfy multiple scenarios.

1.4.3 Values as Strings

As you can see, say, in *aas_core3.types.Property* class, the `value` property holds strings. This is indeed intentional though it might seem a bit outlandish.

You have to bear in mind that the lexical space of XML basic data types, which we use to encode values in such properties, is large, and larger than Python primitive types.

For example, `xs:double`'s can have an arbitrary prefix of zeros (`001234` is a valid `xs:double`).

For another example, `xs:decimal` allows for an arbitrary size and precision. In Python, decimal.Decimal is probably our best bet, but we have to fix the precision up-front. It might well be that our application domain requires later more precision than what we specified at first!

Writing code for a setting where various systems interoperate with mixed application domains is difficult. We wanted to stick to the specification, which mandates XML basic data types, and thus leave the parsing of values up to the users. Thus, we do not restrict the domain where our SDK can be used. The users will know the best what precision and form they need.

1.4.4 No AAS Registry

An AAS Registry is considered an external dependency, since it requires network requests. We left it out-of-scope on purpose as this SDK focuses on the data exchange. Further aas-core projects will work on an AAS registry.

One important consequence of leaving out the registry is that some constraints in the meta-model can not be enforced, as we do not know how to resolve the references.

The full list of omitted constraints is available in the code of aas-core-meta.

1.4.5 No Runtime Type Guards

We do not perform any runtime type-guards for performance reasons. It would be simply too slow to check runtime types at *every* entry point in the library. We assume that the user of the library employs mypy and makes sure the objects passed into the library are correct.

In practice, we intentionally do not check for the following issue stemming from the unexpected types (see issue #436):

```
prop = aas_types.Property(
    id_short="Weight",
    # Note that the value here is an int and not a str!
    # Mypy would have complained.
    value=1,
    value_type=aas_types.DataTypeDefXSD.DOUBLE
)
print(
    json.dumps(
        aas_jsonization.to_jsonable(prop),
        indent=2
    )
)
```

which outputs:

```
{
  "idShort": "Weight",
    "valueType": "xs:double",
    // Note that the value here is a number
    // and not a string - we did not check that
    // the runtime types are correct, and the `json`
    // module simply converted it to a number!
  "value": 1,
  "modelType": "Property"
}
```

1.5 Contributing

1.5.1 Issues

Please report bugs or feature requests by creating GitHub issues.

1.5.2 In Code

If you want to contribute in code, pull requests are welcome!

Please do create a new issue before you dive into coding. It can well be that we already started working on the feature, or that there are upstream or downstream complexities involved which you might not be aware of.

1.5.3 SDK Code Generation

The biggest part of the code has been automatically generated by aas-core-codegen. It probably makes most sense to change the generator rather than add new functionality. However, this needs to be decided on a case-by-case basis.

1.5.4 Test Code Generation

The majority of the unit tests has been automatically generated using the Python scripts in the `dev_scripts/` directory.

To re-generate the test code, run:

```
python dev_scripts/generate_all.py
```

1.5.5 Test Data

The test data is automatically generated by aas-core3.0-testgen, and copied to this repository on every change.

1.5.6 Pre-commit Checks

Before you can run pre-commit checks, you need to all the development dependencies. Run in your virtual environment:

```
pip3 install --editable .
pip3 install -r requirements-dev.txt
```

Now you can execute the checks (from the repository root):

```
python continuous_integration/precommit.py
```

Some of the checks, such as formatting, can be automatically fixed. If you want a self-healing checks, run:

```
python continuous_integration/precommit.py --overwrite
```

1.5.7 Pull Requests

Feature branches. We develop using the feature branches, see this section of the Git book.

If you are a member of the development team, create a feature branch directly within the repository.

Otherwise, if you are a non-member contributor, fork the repository and create the feature branch in your forked repository. See this GitHub tutorial for more guidance.

Branch Prefix. Please prefix the branch with your Github user name (*e.g.,* `mristin/Add-some-feature`).

Continuous Integration. GitHub will run the continuous integration (CI) automatically through GitHub actions. The CI includes running the tests, inspecting the code, re-building the documentation *etc.*

1.5.8 Commit Messages

The commit messages follow the guidelines from https://chris.beams.io/posts/git-commit:

- Separate subject from body with a blank line,
- Limit the subject line to 50 characters,
- Capitalize the subject line,
- Do not end the subject line with a period,
- Use the imperative mood in the subject line,
- Wrap the body at 72 characters, and
- Use the body to explain *what* and *why* (instead of *how*).

1.6 Change Log

1.6.1 1.0.4 (2024-04-16)

The `dataSpecification` field in `EmbeddedDataSpecification` is made optional, according to the book.

1.6.2 1.0.3 (2024-03-22)

- Update to aas-core-meta, codegen, testgen cb28d18, c414f32, 6ff39c260 (#23)

  We propagate the fix from abnf-to-regex related to maximum qualifiers which had been mistakenly represented as exact repetition before.

1.6.3 1.0.2 (2024-03-13)

- Update to aas-core-meta, codegen, testgen 79314c6, 94399e1, e1087880 (#20)

  This patch release brings about the fix for patterns concerning dates and date-times with zone offset *14:00* which previously allowed for a concatenation without a plus sign.

1.6.4 1.0.1 (2024-02-14)

- Test and fix for text attached to end XML elements (#18).

  This patch fixes for the edge case where ElementTree's `XMLPullParser` attaches the text to the end element instead of the start element. Previously, some XML files were wrongly reported as incorrect.

  We do not know what causes this different behavior of the parser, but suspect that it has something to do with the size of the parser's buffer.

1.6.5 1.0.0 (2024-02-02)

This is the first stable release. The release candidates stood the test of time, so we are now confident to publish a stable version.

1.6.6 1.0.0rc3 (2023-09-08)

- Update to aas-core-meta, codegen, testgen 4d7e59e, 18986a0, and 9b43de2e (#12)

  In this version, we fix:

  - Constraints AASc-3a-010 and AASd-131, propagated from aas-core-meta pull requests 281 and 280, respectively.

  We also add the following minor feature:

  - Add `__repr__` to `verification.Error` to facilitate debugging in the downstream clients. Propagated from aas-core-codegen pull request 400.

1.6.7 1.0.0rc2 (2023-06-28)

- Update to aas-core-meta, codegen, testgen 44756fb, 607f65c, bf3720d7 (#7)
  - This is an important patch propagating in particular the following fixes which affected the constraints and their documentation:
    * Pull requests in aas-core-meta 271, 272 and 273 which affect the nullability checks in constraints,
    * Pull request in aas-core-meta 275 which affects the documentation of many constraints.

1.6.8 1.0.0rc1 (2023-03-03)

- Initial version

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W