Schema Salad

Peter Amstutz and Common Workflow Language Project contribu

Apr 10, 2024

CONTENTS

Salad is a schema language for describing JSON or YAML structured linked data documents. Salad schema describes rules for preprocessing, structural validation, and hyperlink checking for documents described by a Salad schema. Salad supports rich data modeling with inheritance, template specialization, object identifiers, object references, documentation generation, code generation, and transformation to RDF. Salad provides a bridge between document and record oriented data modeling and the Semantic Web.

The Schema Salad library is Python 3.8+ only.

CHAPTER

ONE

INSTALLATION

```
pip3 install schema_salad
```

If you intend to use the *schema-salad-tool –codegen=python* feature, please include the *[pycodegen]* extra:

```
pip3 install schema_salad[pycodegen]
```

To install from source:

```
git clone https://github.com/common-workflow-language/schema_salad
cd schema_salad
pip3 install .
# or pip3 install .[pycodegen] if needed
```

CHAPTER

TWO

COMMANDS

Schema salad can be used as a command line tool or imported as a Python module:

```
$ schema-salad-tool
usage: schema-salad-tool [-h] [--rdf-serializer RDF_SERIALIZER] [--skip-schemas]
                         [--strict-foreign-properties] [--print-jsonld-context]
                         [--print-rdfs] [--print-avro] [--print-rdf] [--print-pre]
                         [--print-index] [--print-metadata] [--print-inheritance-dot]
                         [--print-fieldrefs-dot] [--codegen language] [--codegen-target␣
↪CODEGEN_TARGET]
                         [--codegen-examples directory] [--codegen-package dotted.package]
                         [--codegen-copyright copyright_string] [--print-oneline]
                         [--print-doc] [--strict | --non-strict]
                         [--verbose | --quiet | --debug] [--only ONLY] [--redirect REDIRECT]
                         [--brand BRAND] [--brandlink BRANDLINK] [--brandstyle BRANDSTYLE]
                         [--brandinverse] [--primtype PRIMTYPE] [--version]
                         [schema] [document]

$ python
>>> import schema_salad
```

Validate a schema:

```
$ schema-salad-tool myschema.yml
```

Validate a document using a schema:

```
$ schema-salad-tool myschema.yml mydocument.yml
```

Generate HTML documentation:

```
$ schema-salad-tool --print-doc myschema.yml > myschema.html
$ # or
$ schema-salad-doc myschema.yml > myschema.html
```

Get JSON-LD context:

```
$ schema-salad-tool --print-jsonld-context myschema.yml mydocument.yml
```

Convert a document to JSON-LD:

```
$ schema-salad-tool --print-pre myschema.yml mydocument.yml > mydocument.jsonld
```

Generate Python classes for loading/generating documents described by the schema (Requires the *[pycodegen]* extra):

```
$ schema-salad-tool --codegen=python myschema.yml > myschema.py
```

Display inheritance relationship between classes as a graphviz 'dot' file and render as SVG:

```
$ schema-salad-tool --print-inheritance-dot myschema.yml | dot -Tsvg > myschema.svg
```

CHAPTER

THREE

CODEGEN EXAMPLES

The examples in the tables below are helpful to see how to use the output of *schema-salad-tool –codegen* in different languages for loading and/or creating/editing/saving objects.

First set of examples is using the CWL v1.2 schema:

Language	Repository	Serialization Example | Deserialization Example	
Python	https://github.com/common-workflow-language/cwl-utils/	create_cwl_from_objects.py	load_document()
Java	https://github.com/common-workflow-language/cwljava/	(Not yet implemented)	PackedWorkflowClassTest.java
TypeScript	https://github.com/common-workflow-lab/cwl-ts-auto	Creating, editing, and saving CWL docs with TypeScript	Loading CWL documents with TypeScript
.Net	https://github.com/common-workflow-lab/CWLDotNet	Creating, editing, and saving CWL docs with .Net	Loading CWL documents with .Net
C++	https://github.com/common-workflow-lab/cwl-cpp-auto	cwl_output_example.cpp	cwl_input_example.cpp
D	https://github.com/common-workflow-lab/cwl-d-auto	How to use	How to use

Second set of examples is for the Galaxy Workflow Format 2 schema:

Language	Path
Python	https://github.com/galaxyproject/gxformat2/blob/master/gxformat2/schema/v19_09.py
Java	https://github.com/galaxyproject/gxformat2/tree/master/java
TypeScript	https://github.com/galaxyproject/gxformat2/tree/master/typescript

CHAPTER

FOUR

QUICK START

Let's say you have a 'basket' record that can contain items measured either by weight or by count. Here's an example:

```
basket:
  - product: bananas
    price: 0.39
    per: pound
    weight: 1
  - product: cucumbers
    price: 0.79
    per: item
    count: 3
```

We want to validate that all the expected fields are present, the measurement is known, and that "count" cannot be a fractional value. Here is an example schema to do that:

```
- name: Product
  doc: |
    The base type for a product.  This is an abstract type, so it
    can't be used directly, but can be used to define other types.
  type: record
  abstract: true
  fields:
    product: string
    price: float

- name: ByWeight
  doc: |
    A product, sold by weight.  Products may be sold by pound or by
    kilogram.  Weights may be fractional.
  type: record
  extends: Product
  fields:
    per:
      type:
        type: enum
        symbols:
          - pound
          - kilogram
      jsonldPredicate: '#per'
    weight: float
```

(continues on next page)

(continued from previous page)

```
- name: ByCount
  doc: |
    A product, sold by count.  The count must be a integer value.
  type: record
  extends: Product
  fields:
    per:
      type:
        type: enum
        symbols:
          - item
      jsonldPredicate: '#per'
    count: int

- name: Basket
  doc: |
    A basket of products.  The 'documentRoot' field indicates it is a
    valid starting point for a document.  The 'basket' field will
    validate subtypes of 'Product' (ByWeight and ByCount).
  type: record
  documentRoot: true
  fields:
    basket:
      type:
        type: array
        items: Product
```

You can check the schema and document in schema_salad/tests/basket_schema.yml and schema_salad/tests/basket.yml:

```
$ schema-salad-tool basket_schema.yml basket.yml
Document `basket.yml` is valid
```

CHAPTER

FIVE

DOCUMENTATION

See the specification and the metaschema (salad schema for itself). For an example application of Schema Salad see the Common Workflow Language.

CHAPTER

SIX

RATIONALE

The JSON data model is an popular way to represent structured data. It is attractive because of it's relative simplicity and is a natural fit with the standard types of many programming languages. However, this simplicity comes at the cost that basic JSON lacks expressive features useful for working with complex data structures and document formats, such as schemas, object references, and namespaces.

JSON-LD is a W3C standard providing a way to describe how to interpret a JSON document as Linked Data by means of a "context". JSON-LD provides a powerful solution for representing object references and namespaces in JSON based on standard web URIs, but is not itself a schema language. Without a schema providing a well defined structure, it is difficult to process an arbitrary JSON-LD document as idiomatic JSON because there are many ways to express the same data that are logically equivalent but structurally distinct.

Several schema languages exist for describing and validating JSON data, such as JSON Schema and Apache Avro data serialization system, however none understand linked data. As a result, to fully take advantage of JSON-LD to build the next generation of linked data applications, one must maintain separate JSON schema, JSON-LD context, RDF schema, and human documentation, despite significant overlap of content and obvious need for these documents to stay synchronized.

Schema Salad is designed to address this gap. It provides a schema language and processing rules for describing structured JSON content permitting URI resolution and strict document validation. The schema language supports linked data through annotations that describe the linked data interpretation of the content, enables generation of JSON-LD context and RDF schema, and production of RDF triples by applying the JSON-LD context. The schema language also provides for robust support of inline documentation.

6.1 Command Line Options

6.1.1 schema-salad-tool

```
usage: schema-salad-tool [-h] [--rdf-serializer RDF_SERIALIZER]
                         [--skip-schemas] [--strict-foreign-properties]
                         [--print-jsonld-context] [--print-rdfs]
                         [--print-avro] [--print-rdf] [--print-pre]
                         [--print-index] [--print-metadata]
                         [--print-inheritance-dot] [--print-fieldrefs-dot]
                         [--codegen language]
                         [--codegen-target CODEGEN_TARGET]
                         [--codegen-examples directory]
                         [--codegen-package dotted.package]
                         [--codegen-copyright copyright_string]
                         [--codegen-spdx-copyright-text spdx_copyright_text [spdx_
```

(continues on next page)

(continued from previous page)

```
↪copyright_text ...]]
                        [--codegen-spdx-license-identifier spdx_license_identifier]
                        [--codegen-parser-info parser_info] [--print-oneline]
                        [--print-doc] [--strict | --non-strict]
                        [--verbose | --quiet | --debug] [--only ONLY]
                        [--redirect REDIRECT] [--brand BRAND]
                        [--brandlink BRANDLINK] [--brandstyle BRANDSTYLE]
                        [--brandinverse] [--primtype PRIMTYPE] [--version]
                        [schema] [document ...]
```

schema

document

-h, --help
: show this help message and exit

--rdf-serializer <rdf_serializer>
: Output RDF serialization format used by –print-rdf(one of turtle (default), n3, nt, xml)

--skip-schemas
: If specified, ignore $schemas sections.

--strict-foreign-properties
: Strict checking of foreign properties

--print-jsonld-context
: Print JSON-LD context for schema

--print-rdfs
: Print RDF schema

--print-avro
: Print Avro schema

--print-rdf
: Print corresponding RDF graph for document

--print-pre
: Print document after preprocessing

--print-index
: Print node index

--print-metadata
: Print document metadata

--print-inheritance-dot
: Print graphviz file of inheritance

--print-fieldrefs-dot
: Print graphviz file of field refs

--codegen <language>
: Generate classes in target language, currently supported: python, java, typescript, dotnet, cpp, dlang

--codegen-target <codegen_target>

Defaults to sys.stdout for Python/C++/Dlang and ./ for Java/TypeScript/.Net

--codegen-examples <directory>

Directory of example documents for test case generation (Java/TypeScript/.Net/Dlang only).

--codegen-package <dotted.package>

Optional override of the package name which is other derived from the base URL (Java/TypeScript/.Net/Dlang only).

--codegen-copyright <copyright_string>

Optional copyright of the input schema.

--codegen-spdx-copyright-text <spdx_copyright_text>

List of copyright text. Each entry will show up as 'SPDX-FileCopyrightText: ...' (Currently c++ only)

--codegen-spdx-license-identifier <spdx_license_identifier>

Optional spdx license identifier, e.g.: GPL-3.0-only (Currently c++ only)

--codegen-parser-info <parser_info>

Optional parser name which is accessible via resulted parser API (Python and Dlang only)

--print-oneline

Print each error message in oneline

--print-doc

Print HTML schema documentation page

--strict

Strict validation (unrecognized or out of place fields are error)

--non-strict

Lenient validation (ignore unrecognized fields)

--verbose

Default logging

--quiet

Only print warnings and errors.

--debug

Print even more logging

--only <only>

Use with –print-doc, document only listed types

--redirect <redirect>

Use with –print-doc, override default link for type

--brand <brand>

Use with –print-doc, set the 'brand' text in nav bar

--brandlink <brandlink>

Use with –print-doc, set the link for 'brand' in nav bar

--brandstyle <brandstyle>

Use with –print-doc, HTML code to link to an external style sheet

--brandinverse

Use with –print-doc

--primtype <primtype>

Use with –print-doc, link to use for primitive types (string, int etc)

--version, -v

Print version

6.1.2 schema-salad-doc

```
usage: schema-salad-doc [-h] [--only ONLY] [--redirect REDIRECT]
                        [--brand BRAND] [--brandlink BRANDLINK]
                        [--brandstyle BRANDSTYLE] [--brandinverse]
                        [--primtype PRIMTYPE] [--debug]
                        schema
```

schema

-h, --help

show this help message and exit

--only <only>

--redirect <redirect>

--brand <brand>

--brandlink <brandlink>

--brandstyle <brandstyle>

--brandinverse

--primtype <primtype>

--debug

6.2 How to add new types to the local Typeshed

If when running `make mypy` you receive errors about modules that can't be found you may need to add type stubs for new modules to the `mypy-stubs/` directory.

```
stubgen -o mypy-stubs module_name
make mypy
```

Note: the module name is not always the name of the PyPI package (`CacheControl` vs `cachecontrol`).

Stubs are just that, you will still need to annotate whichever functions you call.

Oftentimes it is simpler to comment out imports in the `.pyi` stubs that are not needed yet. The goal is represent the public API, or at least the part we use.

6.3 API Reference

This page contains auto-generated API reference documentation[1].

6.3.1 schema_salad

A schema language for describing JSON or YAML structured linked data documents.

Subpackages

schema_salad.avro

Submodules

schema_salad.avro.schema

Contains the Schema classes.

A schema may be one of:

A record, mapping field names to field value data; An enum, containing one of a small set of symbols; An array of values, all of the same schema; A map of values, all of the same schema; A union of other schemas; A unicode string; A 32-bit signed int; A 64-bit signed long; A 32-bit floating-point float; A 64-bit floating-point double; A boolean; or Null.

Module Contents

Classes

Schema	Base class for all Schema classes.
Name	Class to describe Avro name.
Names	Track name set and default namespace during parsing.
NamedSchema	Named Schemas specified in NAMED_TYPES.
Field	
PrimitiveSchema	Valid primitive types are in PRIMITIVE_TYPES.
EnumSchema	Named Schemas specified in NAMED_TYPES.
ArraySchema	Avro array schema class.
MapSchema	Avro map schema class.
NamedMapSchema	Avro named map schema class.
UnionSchema	Avro union schema class.
NamedUnionSchema	Avro named union schema class.
RecordSchema	Named Schemas specified in NAMED_TYPES.

[1] Created with sphinx-autoapi

Functions

get_other_props(all_props, reserved_props)	Retrieve the non-reserved properties from a dictionary of properties.
make_avsc_object(json_data[, names])	Build Avro Schema from data parsed out of JSON string.
is_subtype(types, existing, new)	Check if a new type specification is compatible with an existing type spec.

Attributes

PRIMITIVE_TYPES
NAMED_TYPES
VALID_TYPES
SCHEMA_RESERVED_PROPS
JsonDataType
AtomicPropType
PropType
PropsType
FIELD_RESERVED_PROPS
VALID_FIELD_SORT_ORDERS

schema_salad.avro.schema.**PRIMITIVE_TYPES = ('null', 'boolean', 'string', 'int', 'long', 'float', 'double')**

schema_salad.avro.schema.**NAMED_TYPES = ('enum', 'record')**

schema_salad.avro.schema.**VALID_TYPES**

schema_salad.avro.schema.**SCHEMA_RESERVED_PROPS = ('type', 'name', 'namespace', 'fields', 'items', 'names', 'symbols', 'values', 'doc')**

schema_salad.avro.schema.**JsonDataType**

schema_salad.avro.schema.**AtomicPropType**

schema_salad.avro.schema.**PropType**

schema_salad.avro.schema.**PropsType**

schema_salad.avro.schema.**FIELD_RESERVED_PROPS = ('default', 'name', 'doc', 'order', 'type')**

schema_salad.avro.schema.**VALID_FIELD_SORT_ORDERS = ('ascending', 'descending', 'ignore')**

exception schema_salad.avro.schema.**AvroException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: *schema_salad.exceptions.SchemaException*

Indicates error with the provided schema definition.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –
- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

exception schema_salad.avro.schema.**SchemaParseException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: *AvroException*

Indicates error with the provided schema definition.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –
- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

class schema_salad.avro.schema.**Schema**(*atype*, *other_props=None*)

Base class for all Schema classes.

Parameters

- **atype** (*str*) –
- **other_props** (*Optional[PropsType]*) –

property props: PropsType

Return type
PropsType

get_prop(*key*)

Parameters
key (*str*) –

Return type
Optional[PropType]

set_prop(*key*, *value*)

Parameters

- **key** (*str*) –
- **value** (*Optional[PropType]*) –

Return type
None

class schema_salad.avro.schema.**Name**(*name_attr=None*, *space_attr=None*, *default_space=None*)

Class to describe Avro name.

Parameters

- **name_attr** (*Optional[str]*) –
- **space_attr** (*Optional[str]*) –
- **default_space** (*Optional[str]*) –

property **fullname**: str | None

Return type
Optional[str]

get_space()

Back out a namespace from full name.

Return type
Optional[str]

class schema_salad.avro.schema.**Names**(*default_namespace=None*)

Track name set and default namespace during parsing.

Parameters
default_namespace (*Optional[str]*) –

has_name(*name_attr*, *space_attr*)

Parameters

- **name_attr** (*str*) –
- **space_attr** (*Optional[str]*) –

Return type
bool

get_name(*name_attr*, *space_attr*)

Fetch the stored schema for the given namespace.

Parameters

- **name_attr** (*str*) –
- **space_attr** (*Optional[str]*) –

Return type
Optional[*NamedSchema*]

add_name(*name_attr*, *space_attr*, *new_schema*)

Add a new schema object to the name set.

Parameters

- **name_attr** (*str*) – name value read in schema
- **space_attr** (*Optional[str]*) – namespace value read in schema.
- **new_schema** (NamedSchema) –

Returns
the Name that was just added.

Return type
Name

class schema_salad.avro.schema.**NamedSchema**(*atype*, *name*, *namespace=None*, *names=None*, *other_props=None*)

Bases: *Schema*

Named Schemas specified in NAMED_TYPES.

Parameters

- **atype** (*str*) –
- **name** (*str*) –
- **namespace** (*Optional[str]*) –
- **names** (*Optional[Names]*) –
- **other_props** (*Optional[PropsType]*) –

property **name**: str

Return type
str

class schema_salad.avro.schema.**Field**(*atype*, *name*, *has_default*, *default=None*, *order=None*, *names=None*, *doc=None*, *other_props=None*)

Parameters

- **atype** (*JsonDataType*) –
- **name** (*str*) –
- **has_default** (*bool*) –
- **default** (*Optional[Any]*) –
- **order** (*Optional[str]*) –
- **names** (*Optional[Names]*) –
- **doc** (*Optional[Union[str, List[str]]]*) –
- **other_props** (*Optional[PropsType]*) –

property **default**: Any | None

Return type
Optional[Any]

get_prop(*key*)

Parameters
key (*str*) –

Return type
Optional[PropType]

set_prop(*key*, *value*)

Parameters

- **key** (*str*) –
- **value** (*Optional[PropType]*) –

Return type
None

class schema_salad.avro.schema.**PrimitiveSchema**(*atype*, *other_props=None*)

Bases: *Schema*

Valid primitive types are in PRIMITIVE_TYPES.

Parameters

- **atype** (*str*) –
- **other_props** (*Optional[PropsType]*) –

class schema_salad.avro.schema.**EnumSchema**(*name*, *namespace*, *symbols*, *names=None*, *doc=None*, *other_props=None*)

Bases: *NamedSchema*

Named Schemas specified in NAMED_TYPES.

Parameters

- **name** (*str*) –
- **namespace** (*Optional[str]*) –
- **symbols** (*List[str]*) –
- **names** (*Optional[Names]*) –
- **doc** (*Optional[Union[str, List[str]]]*) –
- **other_props** (*Optional[PropsType]*) –

property symbols: List[str]

Return type
List[str]

class schema_salad.avro.schema.**ArraySchema**(*items*, *names*, *other_props=None*)

Bases: *Schema*

Avro array schema class.

Parameters

- **items** (*JsonDataType*) –
- **names** (*Names*) –
- **other_props** (*Optional[PropsType]*) –

property items: *Schema*

Avro schema describing the array items' type.

Return type
Schema

class schema_salad.avro.schema.**MapSchema**(*values*, *names*, *other_props=None*)

Bases: *Schema*

Avro map schema class.

Parameters

- **values** (*JsonDataType*) –
- **names** (*Names*) –
- **other_props** (*Optional[PropsType]*) –

property values: *Schema*

Avro schema describing the map values' type.

Return type
Schema

class schema_salad.avro.schema.**NamedMapSchema**(*values*, *names*, *name*, *namespace=None*, *doc=None*, *other_props=None*)

Bases: *NamedSchema*

Avro named map schema class.

Parameters

- **values** (*JsonDataType*) –
- **names** (*Names*) –
- **name** (*str*) –
- **namespace** (*Optional[str]*) –
- **doc** (*Optional[Union[str, List[str]]]*) –
- **other_props** (*Optional[PropsType]*) –

property values: *Schema*

Avro schema describing the map values' type.

Return type
Schema

class schema_salad.avro.schema.**UnionSchema**(*schemas*, *names*)

Bases: *Schema*

Avro union schema class.

Parameters

- **schemas** (*List[JsonDataType]*) –

- **names** (Names) –

property schemas: List[Schema]

Avro schemas composing the Union type.

Return type
List[*Schema*]

class schema_salad.avro.schema.**NamedUnionSchema**(*schemas*, *names*, *name*, *namespace=None*, *doc=None*)

Bases: *NamedSchema*

Avro named union schema class.

Parameters

- **schemas** (*List[JsonDataType]*) –
- **names** (Names) –
- **name** (*str*) –
- **namespace** (*Optional[str]*) –
- **doc** (*Optional[Union[str, List[str]]]*) –

property schemas: List[Schema]

Return type
List[*Schema*]

class schema_salad.avro.schema.**RecordSchema**(*name*, *namespace*, *fields*, *names*, *schema_type='record'*, *doc=None*, *other_props=None*)

Bases: *NamedSchema*

Named Schemas specified in NAMED_TYPES.

Parameters

- **name** (*str*) –
- **namespace** (*Optional[str]*) –
- **fields** (*List[PropsType]*) –
- **names** (Names) –
- **schema_type** (*str*) –
- **doc** (*Optional[Union[str, List[str]]]*) –
- **other_props** (*Optional[PropsType]*) –

property fields: List[Field]

Return type
List[*Field*]

static **make_field_objects**(*field_data*, *names*)

We're going to need to make message parameters too.

Parameters

- **field_data** (*List[PropsType]*) –
- **names** (Names) –

Return type
List[*Field*]

schema_salad.avro.schema.**get_other_props**(*all_props*, *reserved_props*)

Retrieve the non-reserved properties from a dictionary of properties.

Parameters

- **reserved_props** (*Tuple[str, Ellipsis]*) – The set of reserved properties to exclude
- **all_props** (*PropsType*) –

Return type
Optional[PropsType]

schema_salad.avro.schema.**make_avsc_object**(*json_data*, *names=None*)

Build Avro Schema from data parsed out of JSON string.

Parameters

- **names** (*Optional[Names]*) – A Name object (tracks seen names and default space)
- **json_data** (*JsonDataType*) –

Return type
Schema

schema_salad.avro.schema.**is_subtype**(*types*, *existing*, *new*)

Check if a new type specification is compatible with an existing type spec.

Parameters

- **types** (*Dict[str, Any]*) –
- **existing** (*PropType*) –
- **new** (*PropType*) –

Return type
bool

schema_salad.tests

Submodules

schema_salad.tests.conftest

Module Contents

Functions

isolated_cache()	Clear the schema_salad metaschema cache.

schema_salad.tests.conftest.**isolated_cache**()

Clear the schema_salad metaschema cache.

Auto-loaded (see autouse) fixture, loaded per test (function scope). Prevents issues when running multiple tests that load metaschemas multiple times or in parallel (*pytest-parallel*, *pytest-xdist*, etc).

Return type
None

schema_salad.tests.matcher

Module Contents

Classes

JsonDiffMatcher	Raise AssertionError with a readable JSON diff when not __eq__().

Functions

StripYAMLComments(yml)

class schema_salad.tests.matcher.**JsonDiffMatcher**(*expected*)

Raise AssertionError with a readable JSON diff when not __eq__().

Used with assert_called_with() so it's possible for a human to see the differences between expected and actual call arguments that include non-trivial data structures.

Parameters
expected (*Any*) –

__eq__(*actual*)

Return self==value.

Parameters
actual (*Any*) –

Return type
bool

schema_salad.tests.matcher.**StripYAMLComments**(*yml*)

Parameters
yml (*str*) –

Return type
Any

schema_salad.tests.test_avro_names

Avro related tests.

Module Contents

Functions

test_avro_loading()	Confirm conversion of SALAD style names to avro.

schema_salad.tests.test_avro_names.**test_avro_loading**()

Confirm conversion of SALAD style names to avro.

Return type
None

schema_salad.tests.test_cg

Module Contents

Functions

test_load()	
test_err()	
test_include()	
test_import()	
test_import2()	
test_err2()	
test_idmap()	
test_idmap2()	
test_load_pt()	
test_shortname()	Test shortname() function.
metaschema_pre()	Prep-parsed schema for testing.
test_load_metaschema(metaschema_pre)	
test_load_by_yaml_metaschema(metaschema_pre)	
test_load_cwlschema()	

Attributes

maxDiff

schema_salad.tests.test_cg.**test_load**()

Return type
None

schema_salad.tests.test_cg.**test_err**()

Return type
None

schema_salad.tests.test_cg.**test_include**()

Return type
None

schema_salad.tests.test_cg.**test_import**()

Return type
None

schema_salad.tests.test_cg.**maxDiff**

schema_salad.tests.test_cg.**test_import2**()

Return type
None

schema_salad.tests.test_cg.**test_err2**()

Return type
None

schema_salad.tests.test_cg.**test_idmap**()

Return type
None

schema_salad.tests.test_cg.**test_idmap2**()

Return type
None

schema_salad.tests.test_cg.**test_load_pt**()

Return type
None

schema_salad.tests.test_cg.**test_shortname**()
Test shortname() function.

Return type
None

schema_salad.tests.test_cg.**metaschema_pre**()

Prep-parsed schema for testing.

Return type
Any

schema_salad.tests.test_cg.**test_load_metaschema**(*metaschema_pre*)

Parameters
metaschema_pre (*Any*) –

Return type
None

schema_salad.tests.test_cg.**test_load_by_yaml_metaschema**(*metaschema_pre*)

Parameters
metaschema_pre (*Any*) –

Return type
None

schema_salad.tests.test_cg.**test_load_cwlschema**()

Return type
None

schema_salad.tests.test_cli_args

test different sets of command line arguments

Module Contents

Functions

captured_output()
test_version()
test_empty_input()

schema_salad.tests.test_cli_args.**captured_output**()

Return type
Iterator[Tuple[io.StringIO, io.StringIO]]

schema_salad.tests.test_cli_args.**test_version**()

Return type
None

schema_salad.tests.test_cli_args.**test_empty_input**()

Return type
None

schema_salad.tests.test_codegen_errors

Tests of helpful error messages.

Module Contents

Functions

test_error_message1(tmp_path)	
test_error_message2(tmp_path)	
test_error_message4(tmp_path)	
test_error_message5(tmp_path)	
test_error_message6(tmp_path)	
test_error_message7(tmp_path)	
test_error_message8(tmp_path)	
test_error_message9(tmp_path)	
test_error_message10(tmp_path)	
test_error_message11(tmp_path)	
test_error_message15(tmp_path)	
load_document_by_uri(tmp_path, path)	
python_codegen(file_uri, target[, parser_info, package])	

schema_salad.tests.test_codegen_errors.**test_error_message1**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message2**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message4**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message5**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message6**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message7**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message8**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message9**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message10**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message11**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**test_error_message15**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_codegen_errors.**load_document_by_uri**(*tmp_path*, *path*)

Parameters

- **tmp_path** (*pathlib.Path*) –
- **path** (*Union[str, pathlib.Path]*) –

Return type
Any

schema_salad.tests.test_codegen_errors.**python_codegen**(*file_uri*, *target*, *parser_info=None*, *package=None*)

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **parser_info** (*Optional[str]*) –
- **package** (*Optional[str]*) –

Return type
None

schema_salad.tests.test_cpp_codegen

Test C++ code generation.

Module Contents

Functions

test_cwl_cpp_gen(tmp_path)	End to end test of C++ generator using the CWL v1.0 schema.
test_cwl_cpp_generations(tmp_path, filename)	End to end test of C++ generator using small scenarios.
test_cwl_cpp_generations_with_spdx(tmp_path)	End to end test of C++ generator checking for SPDX headers
cpp_codegen(file_uri, target[, spdx_copyright_text, ...])	Help using the C++ code generation function.

schema_salad.tests.test_cpp_codegen.**test_cwl_cpp_gen**(*tmp_path*)

End to end test of C++ generator using the CWL v1.0 schema.

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_cpp_codegen.**test_cwl_cpp_generations**(*tmp_path*, *filename*)

End to end test of C++ generator using small scenarios.

Parameters

- **tmp_path** (*pathlib.Path*) –
- **filename** (*str*) –

Return type
None

schema_salad.tests.test_cpp_codegen.**test_cwl_cpp_generations_with_spdx**(*tmp_path*)

End to end test of C++ generator checking for SPDX headers

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_cpp_codegen.**cpp_codegen**(*file_uri*, *target*, *spdx_copyright_text=None*, *spdx_license_identifier=None*)

Help using the C++ code generation function.

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **spdx_copyright_text** (*Optional[List[str]]*) –
- **spdx_license_identifier** (*Optional[str]*) –

Return type
None

schema_salad.tests.test_cwl11

Ensure codegen-produced parsers accept $schemas directives

run individually as py.test -k test_cwl11

Module Contents

Functions

cwl_v1_2_schema(tmp_path_factory)	
load_cwl(cwl_v1_2_schema, src)	
test_secondaryFiles(cwl_v1_2_schema)	secondaryFiles
test_outputBinding(cwl_v1_2_schema)	secondaryFiles
test_yaml_tab_error(cwl_v1_2_schema)	Tabs in the file.

Attributes

test_dir_name
SchemaType

schema_salad.tests.test_cwl11.**test_dir_name = 'tests/'**

schema_salad.tests.test_cwl11.**SchemaType**

schema_salad.tests.test_cwl11.**cwl_v1_2_schema**(*tmp_path_factory*)

Parameters
tmp_path_factory (*_pytest.tmpdir.TempPathFactory*) –

Return type
Generator[SchemaType, None, None]

schema_salad.tests.test_cwl11.**load_cwl**(*cwl_v1_2_schema*, *src*)

Parameters

- **cwl_v1_2_schema** (*SchemaType*) –
- **src** (*str*) –

Return type
Tuple[Any, Dict[str, Any]]

schema_salad.tests.test_cwl11.**test_secondaryFiles**(*cwl_v1_2_schema*)

secondaryFiles

Parameters
cwl_v1_2_schema (*SchemaType*) –

Return type
None

schema_salad.tests.test_cwl11.**test_outputBinding**(*cwl_v1_2_schema*)

secondaryFiles

Parameters
cwl_v1_2_schema (*SchemaType*) –

Return type
None

schema_salad.tests.test_cwl11.**test_yaml_tab_error**(*cwl_v1_2_schema*)

Tabs in the file.

Parameters
cwl_v1_2_schema (*SchemaType*) –

Return type
None

schema_salad.tests.test_dlang_codegen

Test D code generation.

Module Contents

Functions

test_cwl_dlang_gen(tmp_path)	End to end test of D generator using the CWL v1.0 schema.
dlang_codegen(file_uri, target)	Help using the D code generation function.

schema_salad.tests.test_dlang_codegen.**test_cwl_dlang_gen**(*tmp_path*)

End to end test of D generator using the CWL v1.0 schema.

Parameters

tmp_path (*pathlib.Path*) –

Return type

None

schema_salad.tests.test_dlang_codegen.**dlang_codegen**(*file_uri*, *target*)

Help using the D code generation function.

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –

Return type

None

schema_salad.tests.test_dotnet_codegen

Module Contents

Functions

test_cwl_gen(tmp_path)
test_meta_schema_gen(tmp_path)
test_class_field(tmp_path)
get_data_uri(resource_path)
dotnet_codegen(file_uri, target[, examples])

Attributes

cwl_file_uri
metaschema_file_uri

schema_salad.tests.test_dotnet_codegen.**test_cwl_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_dotnet_codegen.**test_meta_schema_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_dotnet_codegen.**test_class_field**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_dotnet_codegen.**get_data_uri**(*resource_path*)

Parameters
resource_path (*str*) –

Return type
str

schema_salad.tests.test_dotnet_codegen.**cwl_file_uri**

schema_salad.tests.test_dotnet_codegen.**metaschema_file_uri**

schema_salad.tests.test_dotnet_codegen.**dotnet_codegen**(*file_uri*, *target*, *examples=None*)

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **examples** (*Optional[pathlib.Path]*) –

Return type
None

schema_salad.tests.test_errors

Tests of helpful error messages.

Module Contents

Functions

test_errors()	
test_error_message1()	
test_error_message2()	
test_error_message3()	
test_error_message4()	
test_error_message5()	
test_error_message7()	
test_error_message8()	
test_error_message9()	
test_error_message10()	
test_error_message11()	
test_error_message15()	
test_errors_previously_defined_dict_key()	
test_bad_schema()	
test_bad_schema2()	
test_namespaces_type()	Confirm helpful error message when $namespaces is the wrong type.
test_namespaces_undeclared(caplog)	Confirm warning message a namespace is used but not declared.
test_not_a_namespace1(caplog)	Confirm no warning when relative id contains a colon but prefix doesn't look like a namespace.
test_not_a_namespace2(caplog)	Confirm no warning when relative id contains a colon but prefix doesn't look like a namespace.
test_not_a_namespace3(caplog)	Confirm no warning when relative id starts with a colon.
test_schemas_type()	Confirm helpful error message when $schemas is the wrong type.

schema_salad.tests.test_errors.**test_errors**()

Return type
None

schema_salad.tests.test_errors.**test_error_message1**()

Return type
None

schema_salad.tests.test_errors.**test_error_message2**()

Return type
None

schema_salad.tests.test_errors.**test_error_message3**()

Return type
None

schema_salad.tests.test_errors.**test_error_message4**()

Return type
None

schema_salad.tests.test_errors.**test_error_message5**()

Return type
None

schema_salad.tests.test_errors.**test_error_message7**()

Return type
None

schema_salad.tests.test_errors.**test_error_message8**()

Return type
None

schema_salad.tests.test_errors.**test_error_message9**()

Return type
None

schema_salad.tests.test_errors.**test_error_message10**()

Return type
None

schema_salad.tests.test_errors.**test_error_message11**()

Return type
None

schema_salad.tests.test_errors.**test_error_message15**()

Return type
None

schema_salad.tests.test_errors.**test_errors_previously_defined_dict_key**()

Return type
None

schema_salad.tests.test_errors.**test_bad_schema**()

Return type
None

schema_salad.tests.test_errors.**test_bad_schema2**()

Return type
None

schema_salad.tests.test_errors.**test_namespaces_type**()

Confirm helpful error message when $namespaces is the wrong type.

Return type
None

schema_salad.tests.test_errors.**test_namespaces_undeclared**(*caplog*)

Confirm warning message a namespace is used but not declared.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_errors.**test_not_a_namespace1**(*caplog*)

Confirm no warning when relative id contains a colon but prefix doesn't look like a namespace.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_errors.**test_not_a_namespace2**(*caplog*)

Confirm no warning when relative id contains a colon but prefix doesn't look like a namespace.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_errors.**test_not_a_namespace3**(*caplog*)

Confirm no warning when relative id starts with a colon.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_errors.**test_schemas_type**()

Confirm helpful error message when $schemas is the wrong type.

Return type
None

schema_salad.tests.test_examples

Test examples.

Module Contents

Functions

test_schemas()	
test_bad_schemas(caplog)	Test that bad $schemas refs don't stop parsing.
test_skip_bad_schemas(caplog)	Test that (bad) $schemas refs are properly skipped.
test_self_validate()	
test_print_rdf()	Test --print-rdf.
test_print_rdf_invalid_external_ref()	Test --print-rdf when document references unfetchable external schema.
test_print_pre_schema()	Test --print-pre only schema.
test_print_pre()	Test --print-pre.
test_print_schema_index()	Test --print-index only with a schema.
test_print_index()	Test --print-index.
test_print_schema_metadata()	Test --print-metadata only for a schema.
test_print_metadata()	Test --print-metadata.
test_schema_salad_doc_oneline_doc()	Test schema-salad-doc when the 1st type has only a single doc line.
test_avro_regression()	
test_jsonld_ctx()	
test_idmap()	
test_scoped_ref()	
test_examples()	
test_yaml_float_test()	
test_typedsl_ref()	
test_nested_typedsl_ref()	
test_secondaryFile_dsl_ref()	
test_scoped_id()	
test_rdf_datetime()	Affirm that datetime objects can be serialized in makerdf().
test_yaml_datetime()	Affirm that yaml_no_ts prevents the creation of datetime objects.

continues on next page

Table 1 – continued from previous page

test_subscoped_id()	
test_mixin()	
test_fragment()	
test_file_uri()	
test_sourceline()	
test_cmap()	
test_blank_node_id()	
test_can_use_Any()	Test that 'type: Any' can be used
test_nullable_links()	

schema_salad.tests.test_examples.**test_schemas**()

Return type
None

schema_salad.tests.test_examples.**test_bad_schemas**(*caplog*)

Test that bad $schemas refs don't stop parsing.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_examples.**test_skip_bad_schemas**(*caplog*)

Test that (bad) $schemas refs are properly skipped.

Parameters
caplog (*pytest.LogCaptureFixture*) –

Return type
None

schema_salad.tests.test_examples.**test_self_validate**()

Return type
None

schema_salad.tests.test_examples.**test_print_rdf**()

Test –print-rdf.

Return type
None

schema_salad.tests.test_examples.**test_print_rdf_invalid_external_ref**()

Test –print-rdf when document references unfetchable external schema.

Return type
None

schema_salad.tests.test_examples.**test_print_pre_schema**()

Test –print-pre only schema.

Return type
None

schema_salad.tests.test_examples.**test_print_pre**()

Test –print-pre.

Return type
None

schema_salad.tests.test_examples.**test_print_schema_index**()

Test –print-index only with a schema.

Return type
None

schema_salad.tests.test_examples.**test_print_index**()

Test –print-index.

Return type
None

schema_salad.tests.test_examples.**test_print_schema_metadata**()

Test –print-metadata only for a schema.

Return type
None

schema_salad.tests.test_examples.**test_print_metadata**()

Test –print-metadata.

Return type
None

schema_salad.tests.test_examples.**test_schema_salad_doc_oneline_doc**()

Test schema-salad-doc when the 1st type has only a single doc line.

Return type
None

schema_salad.tests.test_examples.**test_avro_regression**()

Return type
None

schema_salad.tests.test_examples.**test_jsonld_ctx**()

Return type
None

schema_salad.tests.test_examples.**test_idmap**()

Return type
None

schema_salad.tests.test_examples.**test_scoped_ref**()

Return type
None

schema_salad.tests.test_examples.**test_examples**()

Return type
None

schema_salad.tests.test_examples.**test_yaml_float_test**()

Return type
None

schema_salad.tests.test_examples.**test_typedsl_ref**()

Return type
None

schema_salad.tests.test_examples.**test_nested_typedsl_ref**()

Return type
None

schema_salad.tests.test_examples.**test_secondaryFile_dsl_ref**()

Return type
None

schema_salad.tests.test_examples.**test_scoped_id**()

Return type
None

schema_salad.tests.test_examples.**test_rdf_datetime**()

Affirm that datetime objects can be serialized in makerdf().

Return type
None

schema_salad.tests.test_examples.**test_yaml_datetime**()

Affirm that yaml_no_ts prevents the creation of datetime objects.

Return type
None

schema_salad.tests.test_examples.**test_subscoped_id**()

Return type
None

schema_salad.tests.test_examples.**test_mixin**()

Return type
None

schema_salad.tests.test_examples.**test_fragment**()

Return type
None

schema_salad.tests.test_examples.**test_file_uri**()

Return type
None

schema_salad.tests.test_examples.**test_sourceline**()

Return type
None

schema_salad.tests.test_examples.**test_cmap**()

Return type
None

schema_salad.tests.test_examples.**test_blank_node_id**()

Return type
None

schema_salad.tests.test_examples.**test_can_use_Any**()

Test that 'type: Any' can be used

Return type
None

schema_salad.tests.test_examples.**test_nullable_links**()

Return type
None

schema_salad.tests.test_fetch

Module Contents

Classes

testFetcher	Fetch resources from URIs.
CWLTestFetcher	Fetch resources from URIs.

Functions

test_fetcher()
test_cache()

class schema_salad.tests.test_fetch.**testFetcher**(*cache*, *session*)

Bases: *schema_salad.fetcher.Fetcher*

Fetch resources from URIs.

Parameters

- **cache** (*schema_salad.utils.CacheType*) –
- **session** (*Optional[requests.sessions.Session]*) –

fetch_text(*url*, *content_types=None*)

Retrieve the given resource as a string.

Parameters

- **url** (*str*) –
- **content_types** (*Optional[List[str]]*) –

Return type
str

check_exists(*url*)

Check if the given resource exists.

Parameters
url (*str*) –

Return type
bool

urljoin(*base*, *url*)

Construct a full ("absolute") URL by combining a "base URL" with another URL.

Parameters

- **base** (*str*) –
- **url** (*str*) –

Return type
str

class schema_salad.tests.test_fetch.**CWLTestFetcher**(*cache*, *session*)

Bases: *schema_salad.fetcher.Fetcher*

Fetch resources from URIs.

Parameters

- **cache** (*schema_salad.utils.CacheType*) –
- **session** (*Optional[requests.sessions.Session]*) –

fetch_text(*url*, *content_types=None*)

Retrieve the given resource as a string.

Parameters

- **url** (*str*) –
- **content_types** (*Optional[List[str]]*) –

Return type
str

check_exists(*url*)

Check if the given resource exists.

Parameters
url (*str*) –

Return type
bool

urljoin(*base*, *url*)

Construct a full ("absolute") URL by combining a "base URL" with another URL.

Parameters

- **base** (*str*) –
- **url** (*str*) –

Return type

str

schema_salad.tests.test_fetch.**test_fetcher**()

Return type

None

schema_salad.tests.test_fetch.**test_cache**()

Return type

None

schema_salad.tests.test_fp

Module Contents

Functions

test_fp()	

schema_salad.tests.test_fp.**test_fp**()

Return type

None

schema_salad.tests.test_java_codegen

Module Contents

Functions

test_cwl_gen(tmp_path)	
test_meta_schema_gen(tmp_path)	
java_codegen(file_uri, target[, examples])	

schema_salad.tests.test_java_codegen.**test_cwl_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_java_codegen.**test_meta_schema_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_java_codegen.**java_codegen**(*file_uri*, *target*, *examples=None*)

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **examples** (*Optional[pathlib.Path]*) –

Return type
None

schema_salad.tests.test_makedoc

Test `schema-salad-doc`.

(also known as `schema-salad-tool --print-doc`)

For convenience, tests are checking exact strings. In the event of changes in the "mistune" package, makedoc.py, or other changes, feel free to modify the test strings as long as the new HTML renders the same way in typical browsers.

Likewise, if the schema-salad metaschema changes and it is missing one or more of the features tested below, then please copy those old features to a new file and update the affected tests to use those new file(s).

Module Contents

Functions

test_schema_salad_inherit_docs()	Test schema-salad-doc when types inherit and override values from parent types.
generate_doc([schema_data])	Avoid error when calling fixture directly.
fixture_metaschema_doc()	Pytest Fixture of the rendered HTML for the metaschema schema.
test_doc_fenced_code_contents_preserved()	Fenced code contents are not interpreted as Markdown definitions and converted into erroneous HTML.
test_doc_headings_target_anchor(metaschema_dc	Doc headers must have an id and section link.
test_doc_render_table_of_contents(metaschema_	The special Table of Contents token must be replaced with a rendered table.
test_plain_links_autolinked(metaschema_doc)	Plan links should be treated as if they were wrapped in angle brackets.
test_embedded_html_unescaped()	Raw HTML shouldn't get escaped.
test_multiline_list_entries_word_spacing(met	Hanging indents in Markdown lists don't lead to wordsmushing.
test_multiline_list_entries_without_indentic	Hanging indents are not required in Markdown lists.
test_detect_changes_in_html(metaschema_doc, tmp_path)	Catch all for changes in HTML output, please adjust if the changes are innocent.

schema_salad.tests.test_makedoc.**test_schema_salad_inherit_docs**()

Test schema-salad-doc when types inherit and override values from parent types.

Return type
None

schema_salad.tests.test_makedoc.**generate_doc**(*schema_data=None*)

Avoid error when calling fixture directly.

Parameters
schema_data (*Optional[str]*) –

Return type
str

schema_salad.tests.test_makedoc.**fixture_metaschema_doc**()

Pytest Fixture of the rendered HTML for the metaschema schema.

Return type
str

schema_salad.tests.test_makedoc.**test_doc_fenced_code_contents_preserved**()

Fenced code contents are not interpreted as Markdown definitions and converted into erroneous HTML.

An example of problem case is when a definition looks like a Markdown list (e.g.: a YAML array). It must not be converted into HTML contents with list tags. However, special characters (e.g.: <, >) must still be escaped, otherwise they will not be correctly rendered within an HTML <pre><code> block.

Return type
None

schema_salad.tests.test_makedoc.**test_doc_headings_target_anchor**(*metaschema_doc*)

Doc headers must have an id and section link.

Parameters
metaschema_doc (*str*) –

Return type
None

schema_salad.tests.test_makedoc.**test_doc_render_table_of_contents**(*metaschema_doc*)

The special Table of Contents token must be replaced with a rendered table.

Parameters
metaschema_doc (*str*) –

Return type
None

schema_salad.tests.test_makedoc.**test_plain_links_autolinked**(*metaschema_doc*)

Plan links should be treated as if they were wrapped in angle brackets.

Parameters
metaschema_doc (*str*) –

Return type
None

schema_salad.tests.test_makedoc.**test_embedded_html_unescaped**()

Raw HTML shouldn't get escaped.

Return type
None

schema_salad.tests.test_makedoc.**test_multiline_list_entries_word_spacing**(*metaschema_doc*)

Hanging indents in Markdown lists don't lead to wordsmushing.

Parameters
metaschema_doc (*str*) –

Return type
None

schema_salad.tests.test_makedoc.**test_multiline_list_entries_without_indention**(*metaschema_doc*)

Hanging indents are not required in Markdown lists.

Parameters
metaschema_doc (*str*) –

Return type
None

schema_salad.tests.test_makedoc.**test_detect_changes_in_html**(*metaschema_doc*, *tmp_path*)

Catch all for changes in HTML output, please adjust if the changes are innocent.

Parameters

- **metaschema_doc** (*str*) –
- **tmp_path** (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_misc

Module Contents

Functions

test_misc()	
test_load_schema_cache()	

schema_salad.tests.test_misc.**test_misc**()

Return type
None

schema_salad.tests.test_misc.**test_load_schema_cache**()

Return type
None

schema_salad.tests.test_pickling

Tests to ensure that mypyc compiled classes are still pickleable.

See https://mypyc.readthedocs.io/en/latest/differences_from_python.html#pickling-and-copying-objects

Module Contents

Functions

test_recordschema_pickle()	Targeted test of pickling a RecordSchema.
test_extend_and_specialize_enums(tmp_path)	

schema_salad.tests.test_pickling.**test_recordschema_pickle**()

Targeted test of pickling a RecordSchema.

Return type
None

schema_salad.tests.test_pickling.**test_extend_and_specialize_enums**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_print_oneline

Module Contents

Functions

test_print_oneline()	
test_print_oneline_for_invalid_yaml()	
test_print_oneline_for_errors_in_the_same_li	
test_print_oneline_for_errors_in_resolve_re	
test_for_invalid_yaml1()	
test_for_invalid_yaml2()	

schema_salad.tests.test_print_oneline.**test_print_oneline**()

Return type
None

schema_salad.tests.test_print_oneline.**test_print_oneline_for_invalid_yaml**()

Return type
None

schema_salad.tests.test_print_oneline.**test_print_oneline_for_errors_in_the_same_line**()

Return type
None

schema_salad.tests.test_print_oneline.**test_print_oneline_for_errors_in_resolve_ref**()

Return type
None

schema_salad.tests.test_print_oneline.**test_for_invalid_yaml1**()

Return type
None

schema_salad.tests.test_print_oneline.**test_for_invalid_yaml2**()

Return type
None

schema_salad.tests.test_python_codegen

Module Contents

Functions

test_safe_identifiers()	Affirm correct construction of identifiers safe for Python.
test_cwl_gen(tmp_path)	
test_meta_schema_gen(tmp_path)	
test_meta_schema_gen_up_to_date(tmp_path)	
test_meta_schema_gen_no_base(tmp_path)	
python_codegen(file_uri, target[, parser_info, package])	
test_default_parser_info(tmp_path)	
test_parser_info(tmp_path)	
test_use_of_package_for_parser_info(tmp_path)	
test_graph_property()	Test the RDFLib Graph representation of the *$schemas* directive.
test_graph_property_cache()	Test that LoadingOptions properly cache the *$schemas* RDFLib Graph representations.
test_graph_property_empty_schema()	Test that an empty RDFLib Graph is returned when not *$schemas* directive is present.

schema_salad.tests.test_python_codegen.**test_safe_identifiers**()

Affirm correct construction of identifiers safe for Python.

Return type
None

schema_salad.tests.test_python_codegen.**test_cwl_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_meta_schema_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_meta_schema_gen_up_to_date**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_meta_schema_gen_no_base**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**python_codegen**(*file_uri*, *target*, *parser_info=None*, *package=None*)

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **parser_info** (*Optional[str]*) –
- **package** (*Optional[str]*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_default_parser_info**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_parser_info**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_use_of_package_for_parser_info**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_python_codegen.**test_graph_property**()

Test the RDFLib Graph representation of the *$schemas* directive.

Return type
None

schema_salad.tests.test_python_codegen.**test_graph_property_cache**()

Test that LoadingOptions properly cache the *$schemas* RDFLib Graph representations.

Return type
None

schema_salad.tests.test_python_codegen.**test_graph_property_empty_schema**()
Test that an empty RDFLib Graph is returned when not *$schemas* directive is present.

Return type
None

schema_salad.tests.test_real_cwl

Checks loading of some real world tools and workflows found in the wild (e.g. dockstore)

run individually as py.test -k tests/test_real_cwl.py

Module Contents

Classes

TestRealWorldCWL

Attributes

test_dir_name

schema_salad.tests.test_real_cwl.**test_dir_name** = 'tests/test_real_cwl/'

class schema_salad.tests.test_real_cwl.**TestRealWorldCWL**

document_loader: *schema_salad.ref_resolver.Loader*

avsc_names: *schema_salad.avro.schema.Names* | *schema_salad.avro.schema.SchemaParseException* | None

schema_metadata: Dict[str, Any] | None

metaschema_loader: *schema_salad.ref_resolver.Loader* | None

classmethod setup_class()

Return type
None

load_cwl(*src*)

Parameters
src (*str*) –

Return type
None

`test_topmed_single_doc()`

TOPMed Variant Calling Pipeline CWL1

Return type
None

`test_h3agatk_WES()`

H3ABioNet GATK Germline Workflow

Return type
None

`test_h3agatk_SNP()`

H3ABioNet SNPs Workflow

Return type
None

`test_icgc_pancan()`

ICGC PanCan

Return type
None

schema_salad.tests.test_ref_resolver

Test the ref_resolver module.

Module Contents

Functions

is_fs_case_sensitive(path)	
tmp_dir_fixture(request)	
test_Loader_initialisation_for_HOME_env_var(	
test_Loader_initialisation_for_TMP_env_var(tı	
*test_Loader_initialisation_with_neither_TMP_*	
test_Loader_initialisation_disable_doc_cache	
test_DefaultFetcher_urljoin_win32(tmp_dir_fixtı	
test_DefaultFetcher_urljoin_linux(tmp_dir_fixtı	
test_import_list()	
test_fetch_inject_id()	
test_attachments()	
test_check_exists_follows_redirects()	
test_resolve_missing_step_id(caplog)	From issue #cwltool/issues/1635. A Workflow with a Step without

schema_salad.tests.test_ref_resolver.**is_fs_case_sensitive**(*path*)

Parameters
path (*str*) –

Return type
bool

schema_salad.tests.test_ref_resolver.**tmp_dir_fixture**(*request*)

Parameters
request (*_pytest.fixtures.FixtureRequest*) –

Return type
str

schema_salad.tests.test_ref_resolver.**test_Loader_initialisation_for_HOME_env_var**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_Loader_initialisation_for_TMP_env_var**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_Loader_initialisation_with_neither_TMP_HOME_set**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_Loader_initialisation_disable_doc_cache**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_DefaultFetcher_urljoin_win32**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_DefaultFetcher_urljoin_linux**(*tmp_dir_fixture*)

Parameters
tmp_dir_fixture (*str*) –

Return type
None

schema_salad.tests.test_ref_resolver.**test_import_list**()

Return type
None

schema_salad.tests.test_ref_resolver.**test_fetch_inject_id**()

Return type
None

schema_salad.tests.test_ref_resolver.**test_attachments**()

Return type
None

schema_salad.tests.test_ref_resolver.**test_check_exists_follows_redirects**()

Return type
None

schema_salad.tests.test_ref_resolver.**test_resolve_missing_step_id**(*caplog*)

From issue #cwltool/issues/1635. A Workflow with a Step without the name attribute must raise a ValidationException that contains the SourceLine data.

Parameters
caplog (*Any*) –

Return type
None

schema_salad.tests.test_schema

Module Contents

Functions

test_extend_and_specialize_enums(tmp_path)	

Attributes

cwl_file_uri	

schema_salad.tests.test_schema.**cwl_file_uri**

schema_salad.tests.test_schema.**test_extend_and_specialize_enums**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_schemas_directive

Checks for accepting $schemas directive

run individually as py.test -k tests/test_schemas_directive.py

Module Contents

Classes

TestSchemasDirective	Ensure codegen-produced parsers accept $schemas directives

Attributes

test_dir_name

schema_salad.tests.test_schemas_directive.**test_dir_name** = 'tests/'

class schema_salad.tests.test_schemas_directive.**TestSchemasDirective**

Ensure codegen-produced parsers accept $schemas directives

document_loader: *schema_salad.ref_resolver.Loader*

avsc_names: *schema_salad.avro.schema.Names* | *schema_salad.avro.schema.SchemaParseException* | None

schema_metadata: Dict[str, Any] | None

metaschema_loader: *schema_salad.ref_resolver.Loader* | None

classmethod setup_class()

Return type
None

load_cwl(*src*)

Parameters
src (*str*) –

Return type
Tuple[Any, Dict[str, Any]]

test_dollarsign_schema()

EDAM.owl as a schema

Return type
None

schema_salad.tests.test_subtypes

Confirm subtypes.

Module Contents

Functions

test_subtypes(old, new, result)	Test is_subtype() function.
test_avro_loading_subtype()	Confirm conversion of SALAD style names to avro when overriding.
test_avro_loading_subtype_bad()	Confirm subtype error when overriding incorrectly.
test_subtypes_nested()	Confirm correct subtype handling on a nested type definition.
test_subtypes_nested_bad()	Confirm subtype error when overriding incorrectly in nested types.
test_subtypes_recursive()	Confirm correct subtype handling on a recursive type definition.
test_subtypes_union()	Confirm correct subtype handling on an union type definition.
test_subtypes_union_bad()	Confirm subtype error when overriding incorrectly in array types.

Attributes

types

schema_salad.tests.test_subtypes.**types = [(['int', 'float', 'double'], 'int', True), (['int', 'float', 'double'], ['int'], True),...**

schema_salad.tests.test_subtypes.**test_subtypes**(*old*, *new*, *result*)

Test is_subtype() function.

Parameters

- **old** (*schema_salad.avro.schema.PropType*) –
- **new** (*schema_salad.avro.schema.PropType*) –
- **result** (*bool*) –

Return type
None

schema_salad.tests.test_subtypes.**test_avro_loading_subtype**()

Confirm conversion of SALAD style names to avro when overriding.

Return type
None

schema_salad.tests.test_subtypes.**test_avro_loading_subtype_bad**()

Confirm subtype error when overriding incorrectly.

Return type
None

schema_salad.tests.test_subtypes.**test_subtypes_nested**()

Confirm correct subtype handling on a nested type definition.

Return type
None

schema_salad.tests.test_subtypes.**test_subtypes_nested_bad**()

Confirm subtype error when overriding incorrectly in nested types.

Return type
None

schema_salad.tests.test_subtypes.**test_subtypes_recursive**()

Confirm correct subtype handling on a recursive type definition.

Return type
None

schema_salad.tests.test_subtypes.**test_subtypes_union**()

Confirm correct subtype handling on an union type definition.

Return type
None

schema_salad.tests.test_subtypes.**test_subtypes_union_bad**()

Confirm subtype error when overriding incorrectly in array types.

Return type
None

schema_salad.tests.test_typescript_codegen

Module Contents

Functions

test_cwl_gen(tmp_path)	
test_meta_schema_gen(tmp_path)	
test_class_field(tmp_path)	
get_data_uri(resource_path)	
typescript_codegen(file_uri, target[, examples])	

Attributes

cwl_file_uri
metaschema_file_uri

schema_salad.tests.test_typescript_codegen.**test_cwl_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_typescript_codegen.**test_meta_schema_gen**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_typescript_codegen.**test_class_field**(*tmp_path*)

Parameters
tmp_path (*pathlib.Path*) –

Return type
None

schema_salad.tests.test_typescript_codegen.**get_data_uri**(*resource_path*)

Parameters
resource_path (*str*) –

Return type
str

schema_salad.tests.test_typescript_codegen.**cwl_file_uri**

schema_salad.tests.test_typescript_codegen.**metaschema_file_uri**

schema_salad.tests.test_typescript_codegen.**typescript_codegen**(*file_uri*, *target*, *examples=None*)

Parameters

- **file_uri** (*str*) –
- **target** (*pathlib.Path*) –
- **examples** (*Optional[pathlib.Path]*) –

Return type
None

schema_salad.tests.util

Shared test functions and attributes.

Module Contents

Functions

get_data(filename)	Get the file path for a given schema file name.
get_data_uri(resource_path)	Get the file URI for tests.

Attributes

cwl_file_uri
metaschema_file_uri
basket_file_uri

schema_salad.tests.util.**get_data**(*filename*)

Get the file path for a given schema file name.

It is able to find file names in the `schema_salad` namespace, but also able to load schema files from the `tests` directory.

Parameters
filename (*str*) –

Return type
Optional[str]

schema_salad.tests.util.**get_data_uri**(*resource_path*)

Get the file URI for tests.

Parameters
resource_path (*str*) –

Return type
str

schema_salad.tests.util.**cwl_file_uri**

schema_salad.tests.util.**metaschema_file_uri**

schema_salad.tests.util.**basket_file_uri**

Submodules

schema_salad.__main__

Default entry point for the schema-salad module.

schema_salad.codegen

Generate language specific loaders for a particular SALAD schema.

Module Contents

Functions

codegen(lang, i, schema_metadata, loader[, target, ...])	Generate classes with loaders for the given Schema Salad description.

Attributes

FIELD_SORT_ORDER

schema_salad.codegen.**FIELD_SORT_ORDER = ['id', 'class', 'name']**

schema_salad.codegen.**codegen**(*lang*, *i*, *schema_metadata*, *loader*, *target=None*, *examples=None*, *package=None*, *copyright=None*, *spdx_copyright_text=None*, *spdx_license_identifier=None*, *parser_info=None*)

Generate classes with loaders for the given Schema Salad description.

Parameters

- **lang** (*str*) –
- **i** (*List[Dict[str, str]]*) –
- **schema_metadata** (*Dict[str, Any]*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **target** (*Optional[str]*) –
- **examples** (*Optional[str]*) –
- **package** (*Optional[str]*) –
- **copyright** (*Optional[str]*) –
- **spdx_copyright_text** (*Optional[List[str]]*) –
- **spdx_license_identifier** (*Optional[str]*) –
- **parser_info** (*Optional[str]*) –

Return type
None

schema_salad.codegen_base

Base class for the generation of loaders from schema-salad definitions.

Module Contents

Classes

TypeDef	Schema Salad type description.
LazyInitDef	Lazy initialization logic.
CodeGenBase	Abstract base class for schema salad code generators.

class schema_salad.codegen_base.**TypeDef**(*name*, *init*, *is_uri=False*, *scoped_id=False*, *ref_scope=0*, *loader_type=None*, *instance_type=None*, *abstract=False*)

Schema Salad type description.

Parameters

- **name** (*str*) –
- **init** (*str*) –
- **is_uri** (*bool*) –
- **scoped_id** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **loader_type** (*Optional[str]*) –
- **instance_type** (*Optional[str]*) –
- **abstract** (*bool*) –

__slots__ = ['name', 'init', 'is_uri', 'scoped_id', 'ref_scope', 'loader_type', 'instance_type', 'abstract']

class schema_salad.codegen_base.**LazyInitDef**(*name*, *init*)

Lazy initialization logic.

Parameters

- **name** (*str*) –
- **init** (*str*) –

__slots__ = ('name', 'init')

class schema_salad.codegen_base.**CodeGenBase**

Abstract base class for schema salad code generators.

declare_type(*declared_type*)

Add this type to our collection, if needed.

Parameters
declared_type (TypeDef) –

Return type
TypeDef

add_lazy_init(*lazy_init*)

Add lazy initialization logic for a given type.

Parameters
lazy_init (LazyInitDef) –

Return type
None

add_vocab(*name*, *uri*)

Add the given name as an abbreviation for the given URI.

Parameters

- **name** (*str*) –
- **uri** (*str*) –

Return type
None

abstract prologue()

Trigger to generate the prolouge code.

Return type
None

abstract static safe_name(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

abstract begin_class(*classname*, *extends*, *doc*, *abstract*, *field_names*, *idfield*, *optional_fields*)

Produce the header for the given class.

Parameters

- **classname** (*str*) –
- **extends** (*MutableSequence[str]*) –
- **doc** (*str*) –
- **abstract** (*bool*) –
- **field_names** (*MutableSequence[str]*) –
- **idfield** (*str*) –
- **optional_fields** (*Set[str]*) –

Return type
None

abstract end_class(*classname*, *field_names*)

Signal that we are done with this class.

Parameters

- **classname** (*str*) –
- **field_names** (*List[str]*) –

Return type
None

abstract type_loader(*type_declaration*, *container=None*, *no_link_check=None*)

Parse the given type declaration and declare its components.

Parameters

- **type_declaration** (*Union[List[Any], Dict[str, Any]]*) –
- **container** (*Optional[str]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
TypeDef

abstract declare_field(*name*, *fieldtype*, *doc*, *optional*, *subscope*)

Output the code to load the given field.

Parameters

- **name** (*str*) –
- **fieldtype** (TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –
- **subscope** (*Optional[str]*) –

Return type
None

abstract declare_id_field(*name*, *fieldtype*, *doc*, *optional*)

Output the code to handle the given ID field.

Parameters

- **name** (*str*) –
- **fieldtype** (TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –

Return type
None

abstract uri_loader(*inner*, *scoped_id*, *vocab_term*, *ref_scope*, *no_link_check=None*)

Construct the TypeDef for the given URI loader.

Parameters

- **inner** (TypeDef) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
TypeDef

abstract idmap_loader(*field*, *inner*, *map_subject*, *map_predicate*)

Construct the TypeDef for the given mapped ID loader.

Parameters

- **field** (*str*) –
- **inner** (TypeDef) –
- **map_subject** (*str*) –
- **map_predicate** (*Optional[str]*) –

Return type
TypeDef

abstract typedsl_loader(*inner*, *ref_scope*)

Construct the TypeDef for the given DSL loader.

Parameters

- **inner** (TypeDef) –
- **ref_scope** (*Optional[int]*) –

Return type
TypeDef

abstract epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (TypeDef) –

Return type
None

abstract secondaryfilesdsl_loader(*inner*)

Construct the TypeDef for secondary files.

Parameters
inner (TypeDef) –

Return type
TypeDef

schema_salad.cpp_codegen

C++17 code generator for a given Schema Salad definition.

Currently only supports emiting YAML from the C++ objects, not yet parsing YAML into C++ objects.

The generated code requires the libyaml-cpp library & headers

To see an example of usage, look at schema_salad/tests/codegen/cwl.cpp which can be combined with the CWL V1.0 schema as shown below:

```
schema-salad-tool --codegen cpp           schema_salad/tests/test_schema/
↪CommonWorkflowLanguage.yml           > cwl_v1_0.h

g++ --std=c++20 -I. -lyaml-cpp schema_salad/tests/codegen/cwl.cpp -o cwl-v1_0-test
./cwl-v1_0-test

# g++ versions older than version 10 may need "--std=c++2a" instead of "--std=c++20"
```

Module Contents

Classes

ClassDefinition	Prototype of a class.
FieldDefinition	Prototype of a single field from a class definition.
MapDefinition	Prototype of a map.
UnionDefinition	Prototype of a union.
EnumDefinition	Prototype of a enum.
CppCodeGen	Generation of C++ code for a given Schema Salad definition.

Functions

q(s)	Put quotes around a string.
replaceKeywords(s)	Rename keywords that are reserved in C++.
safename(name)	Create a C++ safe name.
safename2(name)	Create a namespaced safename.
split_name(s)	Split url name into its components.
split_field(s)	Split field into its components.
isPrimitiveType(v)	Check if v is a primitve type.
hasFieldValue(e, f, v)	Check if e has a field f value.
isRecordSchema(v)	Check if v is of type record schema.
isEnumSchema(v)	Check if v is of type enum schema.
isArray(v)	Check if v is of type array.
pred(i)	Check if v is any of the simple types.
isArraySchema(v)	Check if v is of type array schema.
isMapSchema(v)	Check if v is of type map schema.
isUnionSchema(v)	Check if v is of type union schema.

schema_salad.cpp_codegen.**q**(*s*)

Put quotes around a string.

Parameters
s (*str*) –

Return type
str

schema_salad.cpp_codegen.**replaceKeywords**(*s*)

Rename keywords that are reserved in C++.

Parameters
s (*str*) –

Return type
str

schema_salad.cpp_codegen.**safename**(*name*)

Create a C++ safe name.

Parameters
name (*str*) –

Return type
str

schema_salad.cpp_codegen.**safename2**(*name*)

Create a namespaced safename.

Parameters
name (*Dict[str, str]*) –

Return type
str

schema_salad.cpp_codegen.**split_name**(*s*)

Split url name into its components.

Splits names like https://xyz.xyz/blub#cwl/class into its class path and non class path

Parameters
s (*str*) –

Return type
Tuple[str, str]

schema_salad.cpp_codegen.**split_field**(*s*)

Split field into its components.

similar to split_name but for field names

Parameters
s (*str*) –

Return type
Tuple[str, str, str]

class schema_salad.cpp_codegen.**ClassDefinition**(*name*)

Prototype of a class.

Parameters
name (*str*) –

writeFwdDeclaration(*target*, *fullInd*, *ind*)

Write forward declaration.

Parameters

- **target** (*IO[str]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

writeDefinition(*target*, *fullInd*, *ind*)

Write definition of the class.

Parameters

- **target** (*IO[Any]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

writeImplDefinition(*target*, *fullInd*, *ind*)

Write definition with implementation.

Parameters

- **target** (*IO[str]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

class schema_salad.cpp_codegen.**FieldDefinition**(*name*, *typeStr*, *optional*, *remap*)

Prototype of a single field from a class definition.

Parameters

- **name** (*str*) –
- **typeStr** (*str*) –
- **optional** (*bool*) –
- **remap** (*str*) –

writeDefinition(*target*, *fullInd*, *ind*, *namespace*)

Write a C++ definition for the class field.

Parameters

- **target** (*IO[Any]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –
- **namespace** (*str*) –

Return type
None

class schema_salad.cpp_codegen.**MapDefinition**(*name*, *values*)

Prototype of a map.

Parameters

- **name** (*str*) –
- **values** (*List[str]*) –

writeFwdDeclaration(*target*, *fullInd*, *ind*)

Write forward declaration.

Parameters

- **target** (*IO[str]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

writeDefinition(*target*, *ind*)

Write map definition to output.

Parameters

- **target** (*IO[str]*) –
- **ind** (*str*) –

Return type
None

writeImplDefinition(*target*, *fullInd*, *ind*)

Write definition with implementation.

Parameters

- **target** (*IO[str]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

class schema_salad.cpp_codegen.**UnionDefinition**(*name*, *types*)

Prototype of a union.

Parameters

- **name** (*str*) –
- **types** (*List[str]*) –

writeFwdDeclaration(*target*, *fullInd*, *ind*)

Write forward declaration.

Parameters

- **target** (*IO[str]*) –

- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

writeDefinition(*target*, *ind*)

Write union definition to output.

Parameters

- **target** (*IO[str]*) –
- **ind** (*str*) –

Return type
None

writeImplDefinition(*target*, *fullInd*, *ind*)

Write definition with implementation.

Parameters

- **target** (*IO[str]*) –
- **fullInd** (*str*) –
- **ind** (*str*) –

Return type
None

class schema_salad.cpp_codegen.**EnumDefinition**(*name*, *values*)

Prototype of a enum.

Parameters

- **name** (*str*) –
- **values** (*List[str]*) –

writeDefinition(*target*, *ind*)

Write enum definition to output.

Parameters

- **target** (*IO[str]*) –
- **ind** (*str*) –

Return type
None

schema_salad.cpp_codegen.**isPrimitiveType**(*v*)

Check if v is a primitve type.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**hasFieldValue**(*e*, *f*, *v*)

Check if e has a field f value.

Parameters

- **e** (*Any*) –
- **f** (*str*) –
- **v** (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isRecordSchema**(*v*)

Check if v is of type record schema.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isEnumSchema**(*v*)

Check if v is of type enum schema.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isArray**(*v*)

Check if v is of type array.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**pred**(*i*)

Check if v is any of the simple types.

Parameters
i (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isArraySchema**(*v*)

Check if v is of type array schema.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isMapSchema**(*v*)

Check if v is of type map schema.

Parameters
v (*Any*) –

Return type
bool

schema_salad.cpp_codegen.**isUnionSchema**(*v*)

Check if v is of type union schema.

Parameters
v (*Any*) –

Return type
bool

class schema_salad.cpp_codegen.**CppCodeGen**(*base*, *target*, *examples*, *package*, *copyright*, *spdx_copyright_text*, *spdx_license_identifier*)

Bases: *schema_salad.codegen_base.CodeGenBase*

Generation of C++ code for a given Schema Salad definition.

Parameters

- **base** (*str*) –
- **target** (*IO[str]*) –
- **examples** (*Optional[str]*) –
- **package** (*str*) –
- **copyright** (*Optional[str]*) –
- **spdx_copyright_text** (*Optional[List[str]]*) –
- **spdx_license_identifier** (*Optional[str]*) –

convertTypeToCpp(*type_declaration*)

Convert a Schema Salad type to a C++ type.

Parameters
type_declaration (*Union[List[Any], Dict[str, Any], str]*) –

Return type
str

epilogue(*root_loader*)

Generate final part of our cpp file.

Parameters
root_loader (*Optional[schema_salad.codegen_base.TypeDef]*) –

Return type
None

parseRecordField(*field*)

Parse a record field.

Parameters
field (*Dict[str, Any]*) –

Return type
FieldDefinition

parseRecordSchema(*stype*)

Parse a record schema.

Parameters
stype (*Dict[str, Any]*) –

Return type
None

parseMapSchema(*stype*)

Parse a map schema.

Parameters
stype (*Dict[str, Any]*) –

Return type
str

parseUnionSchema(*stype*)

Parse a union schema.

Parameters
stype (*Dict[str, Any]*) –

Return type
str

parseEnum(*stype*)

Parse a schema salad enum.

Parameters
stype (*Dict[str, Any]*) –

Return type
str

parse(*items*)

Parse sechema salad items.

This function is being called from the outside and drives the whole code generation.

Parameters
items (*List[Dict[str, Any]]*) –

Return type
None

schema_salad.dlang_codegen

D code generator for a given schema salad definition.

Module Contents

Classes

DlangCodeGen	Generation of D code for a given Schema Salad definition.

class schema_salad.dlang_codegen.**DlangCodeGen**(*base*, *target*, *examples*, *package*, *copyright_*, *parser_info*, *salad_version*)

Bases: *schema_salad.codegen_base.CodeGenBase*

Generation of D code for a given Schema Salad definition.

Parameters

- **base** (*str*) –
- **target** (*IO[str]*) –
- **examples** (*Optional[str]*) –
- **package** (*str*) –
- **copyright_** (*Optional[str]*) –
- **parser_info** (*Optional[str]*) –
- **salad_version** (*str*) –

prologue()

Trigger to generate the prolouge code.

Return type
None

epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (schema_salad.codegen_base.TypeDef) –

Return type
None

static safe_name(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

to_doc_comment(*doc*)

Return an embedded documentation comments for a given string.

Parameters
doc (*Union[None, str, List[str]]*) –

Return type
str

parse_record_field_type(*type_*, *jsonld_pred*)

Return an annotation string and a type string.

Parameters

- **type_** (*Any*) –
- **jsonld_pred** (*Union[None, str, Dict[str, Any]]*) –

Return type
Tuple[str, str]

parse_record_field(*field*, *parent_name=None*)

Return a declaration string for a given record field.

Parameters

- **field** (*Dict[str, Any]*) –
- **parent_name** (*Optional[str]*) –

Return type
str

parse_record_schema(*stype*)

Return a declaration string for a given record schema.

Parameters
stype (*Dict[str, Any]*) –

Return type
str

parse_enum(*stype*)

Return a declaration string for a given enum schema.

Parameters
stype (*Dict[str, Any]*) –

Return type
str

parse(*items*)

Generate D code from items and write it to target.

Parameters
items (*List[Dict[str, Any]]*) –

Return type
None

schema_salad.dotnet_codegen

DotNet code generator for a given schema salad definition.

Module Contents

Classes

DotNetCodeGen	Generation of TypeScript code for a given Schema Salad definition.

Functions

doc_to_doc_string(doc[, indent_level])	Generate a documentation string from a schema salad doc field.

Attributes

prims

schema_salad.dotnet_codegen.**doc_to_doc_string**(*doc*, *indent_level=0*)

Generate a documentation string from a schema salad doc field.

Parameters

- **doc** (*Optional[str]*) –
- **indent_level** (*int*) –

Return type

str

schema_salad.dotnet_codegen.**prims**

class schema_salad.dotnet_codegen.**DotNetCodeGen**(*base*, *examples*, *target*, *package*)

Bases: *schema_salad.codegen_base.CodeGenBase*

Generation of TypeScript code for a given Schema Salad definition.

Parameters

- **base** (*str*) –
- **examples** (*Optional[str]*) –
- **target** (*Optional[str]*) –
- **package** (*str*) –

prologue()

Trigger to generate the prolouge code.

Return type
None

static safe_name(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

begin_class(*classname*, *extends*, *doc*, *abstract*, *field_names*, *idfield*, *optional_fields*)

Produce the header for the given class.

Parameters

- **classname** (*str*) –
- **extends** (*MutableSequence[str]*) –
- **doc** (*str*) –
- **abstract** (*bool*) –
- **field_names** (*MutableSequence[str]*) –
- **idfield** (*str*) –
- **optional_fields** (*Set[str]*) –

Return type
None

end_class(*classname*, *field_names*)

Signal that we are done with this class.

Parameters

- **classname** (*str*) –
- **field_names** (*List[str]*) –

Return type
None

type_loader(*type_declaration*, *container=None*, *no_link_check=None*)

Parse the given type declaration and declare its components.

Parameters

- **type_declaration** (*Union[List[Any], Dict[str, Any], str]*) –
- **container** (*Optional[str]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

type_loader_enum(*type_declaration*)

Parameters
type_declaration (*Dict[str, Any]*) –

Return type
schema_salad.codegen_base.TypeDef

declare_field(*name*, *fieldtype*, *doc*, *optional*, *subscope*)

Output the code to load the given field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –
- **subscope** (*Optional[str]*) –

Return type
None

declare_id_field(*name*, *fieldtype*, *doc*, *optional*)

Output the code to handle the given ID field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –

Return type
None

to_dotnet(*val*)

Convert a Python keyword to a DotNet keyword.

Parameters
val (*Any*) –

Return type
Any

uri_loader(*inner*, *scoped_id*, *vocab_term*, *ref_scope*, *no_link_check=None*)

Construct the TypeDef for the given URI loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

idmap_loader(*field*, *inner*, *map_subject*, *map_predicate*)

Construct the TypeDef for the given mapped ID loader.

Parameters

- **field** (*str*) –
- **inner** (schema_salad.codegen_base.TypeDef) –
- **map_subject** (*str*) –
- **map_predicate** (*Optional[str]*) –

Return type
schema_salad.codegen_base.TypeDef

typedsl_loader(*inner*, *ref_scope*)

Construct the TypeDef for the given DSL loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **ref_scope** (*Optional[int]*) –

Return type
schema_salad.codegen_base.TypeDef

epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (schema_salad.codegen_base.TypeDef) –

Return type
None

secondaryfilesdsl_loader(*inner*)

Construct the TypeDef for secondary files.

Parameters
inner (schema_salad.codegen_base.TypeDef) –

Return type
schema_salad.codegen_base.TypeDef

schema_salad.exceptions

Shared Exception classes.

Module Contents

Functions

to_one_line_messages(exc)

exception schema_salad.exceptions.**SchemaSaladException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: Exception

Base class for all schema-salad exceptions.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –
- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

propagate_sourceline()

Return type
None

as_warning()

Return type
SchemaSaladException

with_sourceline(*sl*)

Parameters
sl (*Optional[schema_salad.sourceline.SourceLine]*) –

Return type
SchemaSaladException

leaves()

Return type
List[*SchemaSaladException*]

prefix()

Return type
str

summary(*level=0*, *with_bullet=False*)

Parameters

- **level** (*int*) –
- **with_bullet** (*bool*) –

Return type
str

__str__()
Convert to a string using *pretty_str()*.

Return type
str

pretty_str(*level=0*)

Parameters
level (*int*) –

Return type
str

exception schema_salad.exceptions.**SchemaException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: *SchemaSaladException*

Indicates error with the provided schema definition.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –
- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

exception schema_salad.exceptions.**ValidationException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: *SchemaSaladException*

Indicates error with document against the provided schema.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –
- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

exception schema_salad.exceptions.**ClassValidationException**(*msg*, *sl=None*, *children=None*, *bullet_for_children=''*)

Bases: *ValidationException*

Indicates error with document against the provided schema.

Parameters

- **msg** (*str*) –
- **sl** (*Optional[schema_salad.sourceline.SourceLine]*) –

- **children** (*Optional[Sequence[SchemaSaladException]]*) –
- **bullet_for_children** (*str*) –

schema_salad.exceptions.**to_one_line_messages**(*exc*)

Parameters
exc (SchemaSaladException) –

Return type
str

schema_salad.fetcher

Resource fetching.

Module Contents

Classes

Fetcher	Fetch resources from URIs.
MemoryCachingFetcher	Fetcher that caches resources in memory after retrieval.
DefaultFetcher	The default Fetcher implementation.

class schema_salad.fetcher.**Fetcher**

Bases: abc.ABC

Fetch resources from URIs.

schemes = ['file', 'http', 'https', 'mailto']

abstract fetch_text(*url*, *content_types=None*)

Retrieve the given resource as a string.

Parameters

- **url** (*str*) –
- **content_types** (*Optional[List[str]]*) –

Return type
str

abstract check_exists(*url*)

Check if the given resource exists.

Parameters
url (*str*) –

Return type
bool

abstract urljoin(*base_url*, *url*)

Construct a full ("absolute") URL by combining a "base URL" with another URL.

Parameters

- **base_url** (*str*) –
- **url** (*str*) –

Return type
str

supported_schemes()

Return the list of supported URI schemes.

Return type
List[str]

class schema_salad.fetcher.**MemoryCachingFetcher**(*cache*)

Bases: *Fetcher*

Fetcher that caches resources in memory after retrieval.

Parameters
cache (*schema_salad.utils.CacheType*) –

class schema_salad.fetcher.**DefaultFetcher**(*cache*, *session*)

Bases: *MemoryCachingFetcher*

The default Fetcher implementation.

Parameters

- **cache** (*schema_salad.utils.CacheType*) –
- **session** (*Optional[requests.sessions.Session]*) –

fetch_text(*url*, *content_types=None*)

Retrieve the given resource as a string.

Parameters

- **url** (*str*) –
- **content_types** (*Optional[List[str]]*) –

Return type
str

check_exists(*url*)

Check if the given resource exists.

Parameters
url (*str*) –

Return type
bool

urljoin(*base_url*, *url*)

Construct a full ("absolute") URL by combining a "base URL" with another URL.

Parameters

- **base_url** (*str*) –
- **url** (*str*) –

Return type

str

schema_salad.java_codegen

Java code generator for a given schema salad definition.

Module Contents

Classes

JavaCodeGen	Abstract base class for schema salad code generators.

Functions

doc_to_doc_string(doc[, indent_level])

Attributes

USE_ONE_OR_LIST_OF_TYPES
BASIC_JAVA_IDENTIFIER_RE
prims

schema_salad.java_codegen.**USE_ONE_OR_LIST_OF_TYPES = False**

schema_salad.java_codegen.**BASIC_JAVA_IDENTIFIER_RE**

schema_salad.java_codegen.**doc_to_doc_string**(*doc*, *indent_level=0*)

Parameters

- **doc** (*Optional[str]*) –
- **indent_level** (*int*) –

Return type
str

schema_salad.java_codegen.**prims**

class schema_salad.java_codegen.**JavaCodeGen**(*base*, *target*, *examples*, *package*, *copyright*)

Bases: *schema_salad.codegen_base.CodeGenBase*

Abstract base class for schema salad code generators.

Parameters

- **base** (*str*) –
- **target** (*Optional[str]*) –
- **examples** (*Optional[str]*) –
- **package** (*str*) –
- **copyright** (*Optional[str]*) –

prologue()

Trigger to generate the prolouge code.

Return type
None

static property_name(*name*)

Parameters
name (*str*) –

Return type
str

static safe_name(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

interface_name(*n*)

Parameters
n (*str*) –

Return type
str

begin_class(*classname*, *extends*, *doc*, *abstract*, *field_names*, *idfield*, *optional_fields*)

Produce the header for the given class.

Parameters

- **classname** (*str*) –
- **extends** (*MutableSequence[str]*) –
- **doc** (*str*) –

- **abstract** (*bool*) –
- **field_names** (*MutableSequence[str]*) –
- **idfield** (*str*) –
- **optional_fields** (*Set[str]*) –

Return type
None

end_class(*classname*, *field_names*)

Finish this class.

Parameters

- **classname** (*str*) –
- **field_names** (*List[str]*) –

Return type
None

type_loader(*type_declaration*, *container=None*, *no_link_check=None*)

Parse the given type declaration and declare its components.

Parameters

- **type_declaration** (*Union[List[Any], Dict[str, Any], str]*) –
- **container** (*Optional[str]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

type_loader_enum(*type_declaration*)

Parameters
type_declaration (*Dict[str, Any]*) –

Return type
schema_salad.codegen_base.TypeDef

declare_field(*name*, *fieldtype*, *doc*, *optional*, *subscope*)

Output the code to load the given field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –
- **subscope** (*Optional[str]*) –

Return type
None

declare_id_field(*name*, *fieldtype*, *doc*, *optional*)

Output the code to handle the given ID field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –

Return type
None

uri_loader(*inner*, *scoped_id*, *vocab_term*, *ref_scope*, *no_link_check=None*)

Construct the TypeDef for the given URI loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

idmap_loader(*field*, *inner*, *map_subject*, *map_predicate*)

Construct the TypeDef for the given mapped ID loader.

Parameters

- **field** (*str*) –
- **inner** (schema_salad.codegen_base.TypeDef) –
- **map_subject** (*str*) –
- **map_predicate** (*Optional[str]*) –

Return type
schema_salad.codegen_base.TypeDef

typedsl_loader(*inner*, *ref_scope*)

Construct the TypeDef for the given DSL loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **ref_scope** (*Union[int, None]*) –

Return type
schema_salad.codegen_base.TypeDef

to_java(*val*)

Parameters
val (*Any*) –

Return type
Any

epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (schema_salad.codegen_base.TypeDef) –

Return type
None

secondaryfilesdsl_loader(*inner*)

Construct the TypeDef for secondary files.

Parameters
inner (schema_salad.codegen_base.TypeDef) –

Return type
schema_salad.codegen_base.TypeDef

schema_salad.jsonld_context

Module Contents

Functions

pred(datatype, field, name, context, defaultBase, ...)	
process_type(t, g, context, defaultBase, namespaces, ...)	
salad_to_jsonld_context(j, schema_ctx)	
fix_jsonld_ids(obj, ids)	Add missing identity entries.
makerdf(workflow, wf, ctx[, graph])	

schema_salad.jsonld_context.**pred**(*datatype*, *field*, *name*, *context*, *defaultBase*, *namespaces*)

Parameters

- **datatype** (*MutableMapping[str, Union[Dict[str, str], str]]*) –
- **field** (*Optional[Dict[str, Any]]*) –
- **name** (*str*) –
- **context** (*schema_salad.utils.ContextType*) –
- **defaultBase** (*str*) –
- **namespaces** (*Dict[str, rdflib.namespace.Namespace]*) –

Return type
Union[Dict[str, Optional[str]], str]

schema_salad.jsonld_context.**process_type**(*t*, *g*, *context*, *defaultBase*, *namespaces*, *defaultPrefix*)

Parameters

- **t** (*MutableMapping[str, Any]*) –

- **g** (*rdflib.Graph*) –
- **context** (*schema_salad.utils.ContextType*) –
- **defaultBase** (*str*) –
- **namespaces** (*Dict[str, rdflib.namespace.Namespace]*) –
- **defaultPrefix** (*str*) –

Return type
None

schema_salad.jsonld_context.**salad_to_jsonld_context**(*j*, *schema_ctx*)

Parameters

- **j** (*Iterable[MutableMapping[str, Any]]*) –
- **schema_ctx** (*MutableMapping[str, Any]*) –

Return type
Tuple[schema_salad.utils.ContextType, rdflib.Graph]

schema_salad.jsonld_context.**fix_jsonld_ids**(*obj*, *ids*)

Add missing identity entries.

Parameters

- **obj** (*Union[ruamel.yaml.comments.CommentedMap, float, str, ruamel.yaml.comments.CommentedSeq]*) –
- **ids** (*List[str]*) –

Return type
None

schema_salad.jsonld_context.**makerdf**(*workflow*, *wf*, *ctx*, *graph=None*)

Parameters

- **workflow** (*Optional[str]*) –
- **wf** (*Union[ruamel.yaml.comments.CommentedMap, float, str, ruamel.yaml.comments.CommentedSeq]*) –
- **ctx** (*schema_salad.utils.ContextType*) –
- **graph** (*Optional[rdflib.Graph]*) –

Return type
rdflib.Graph

schema_salad.main

Command line interface to schema-salad.

Module Contents

Functions

printrdf(workflow, wf, ctx, sr)	
arg_parser()	Build the argument parser.
main([argsl])	

schema_salad.main.**printrdf**(*workflow*, *wf*, *ctx*, *sr*)

Parameters

- **workflow** (*str*) –
- **wf** (*Union[ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq]*) –
- **ctx** (*Dict[str, Any]*) –
- **sr** (*str*) –

Return type
None

schema_salad.main.**arg_parser**()

Build the argument parser.

Return type
argparse.ArgumentParser

schema_salad.main.**main**(*argsl=None*)

Parameters
argsl (*Optional[List[str]]*) –

Return type
int

schema_salad.makedoc

Module Contents

Classes

MyRenderer	Custom renderer with different representations of selected HTML tags.
ToC	
RenderType	

Functions

escape_html(s)	Escape HTML but otherwise preserve single quotes.
vocab_type_name(url)	Remove the avro namespace, if any.
has_types(items)	
linkto(item)	
patch_fenced_code(original_markdown_text, ...)	Reverts fenced code fragments found in the modified contents back to their original definition.
to_id(text)	
number_headings(toc, maindoc)	
fix_doc(doc)	
avrold_doc(j, outdoc, renderlist, redirects, brand, ...)	
arg_parser()	Build the argument parser.
main()	Shortcut entrypoint.
makedoc(stdout, schema[, redirects, only, brand, ...])	Emit HTML representation of a given schema.

Attributes

PluginName	
basicTypes	

schema_salad.makedoc.**PluginName**

schema_salad.makedoc.**escape_html**(*s*)

Escape HTML but otherwise preserve single quotes.

Parameters
s (*str*) –

Return type
str

schema_salad.makedoc.**vocab_type_name**(*url*)

Remove the avro namespace, if any.

Parameters
url (*str*) –

Return type
str

schema_salad.makedoc.**has_types**(*items*)

Parameters
items (*Any*) –

Return type
List[str]

schema_salad.makedoc.**linkto**(*item*)

Parameters
item (*str*) –

Return type
str

class schema_salad.makedoc.**MyRenderer**(*escape=True*, *allow_harmful_protocols=None*)

Bases: mistune.renderers.html.HTMLRenderer

Custom renderer with different representations of selected HTML tags.

heading(*text*, *level*, ***attrs*)

Override HTML heading creation with text IDs.

Parameters
- **text** (*str*) –
- **level** (*int*) –
- **attrs** (*Any*) –

Return type
str

text(*text*)

Don't escape quotation marks.

Parameters
text (*str*) –

Return type
str

inline_html(*html*)

Don't escape characters in predefined HTML within paragraph tags.

Parameters
html (*str*) –

Return type
str

block_html(*html*)

Don't escape characters nor wrap predefined HTML within paragraph tags.

Parameters
html (*str*) –

Return type
str

block_code(*code*, *info=None*)

Don't escape quotation marks.

Parameters

- **code** (*str*) –
- **info** (*Optional[str]*) –

Return type
str

schema_salad.makedoc.**patch_fenced_code**(*original_markdown_text*, *modified_markdown_text*)

Reverts fenced code fragments found in the modified contents back to their original definition.

Parameters

- **original_markdown_text** (*str*) –
- **modified_markdown_text** (*str*) –

Return type
str

schema_salad.makedoc.**to_id**(*text*)

Parameters
text (*str*) –

Return type
str

class schema_salad.makedoc.**ToC**

add_entry(*thisdepth*, *title*)

Add an entry to the table of contents.

Parameters

- **thisdepth** (*int*) –
- **title** (*str*) –

Return type
str

contents(*idn*)

Parameters
idn (*str*) –

Return type
str

schema_salad.makedoc.**basicTypes = ('https://w3id.org/cwl/salad#null', 'http://www.w3.org/2001/XMLSchema#boolean',...**

schema_salad.makedoc.**number_headings**(*toc*, *maindoc*)

Parameters

- **toc** (ToC) –
- **maindoc** (*str*) –

Return type
str

schema_salad.makedoc.**fix_doc**(*doc*)

Parameters
doc (*Union[List[str], str]*) –

Return type
str

class schema_salad.makedoc.**RenderType**(*toc*, *j*, *renderlist*, *redirects*, *primitiveType*)

Parameters

- **toc** (ToC) –
- **j** (*List[Dict[str, Any]]*) –
- **renderlist** (*List[str]*) –
- **redirects** (*Dict[str, str]*) –
- **primitiveType** (*str*) –

typefmt(*tp*, *redirects*, *nbsp=False*, *jsonldPredicate=None*)

Parameters

- **tp** (*Any*) –
- **redirects** (*Dict[str, str]*) –
- **nbsp** (*bool*) –
- **jsonldPredicate** (*Optional[Union[Dict[str, str], str]]*) –

Return type
str

render_type(*f*, *depth*)

Parameters

- **f** (*Dict[str, Any]*) –
- **depth** (*int*) –

Return type
None

schema_salad.makedoc.**avrold_doc**(*j*, *outdoc*, *renderlist*, *redirects*, *brand*, *brandlink*, *primtype*, *brandstyle=None*, *brandinverse=False*)

Parameters

- **j** (*List[Dict[str, Any]]*) –
- **outdoc** (*IO[Any]*) –
- **renderlist** (*List[str]*) –
- **redirects** (*Dict[str, str]*) –
- **brand** (*str*) –
- **brandlink** (*str*) –
- **primtype** (*str*) –
- **brandstyle** (*Optional[str]*) –

- **brandinverse** (*Optional[bool]*) –

Return type
None

schema_salad.makedoc.**arg_parser**()

Build the argument parser.

Return type
argparse.ArgumentParser

schema_salad.makedoc.**main**()

Shortcut entrypoint.

Return type
None

schema_salad.makedoc.**makedoc**(*stdout*, *schema*, *redirects=None*, *only=None*, *brand=None*, *brandlink=None*, *primtype=None*, *brandstyle=None*, *brandinverse=False*)

Emit HTML representation of a given schema.

Parameters

- **stdout** (*IO[Any]*) –
- **schema** (*str*) –
- **redirects** (*Optional[List[str]]*) –
- **only** (*Optional[List[str]]*) –
- **brand** (*Optional[str]*) –
- **brandlink** (*Optional[str]*) –
- **primtype** (*Optional[str]*) –
- **brandstyle** (*Optional[str]*) –
- **brandinverse** (*Optional[bool]*) –

Return type
None

schema_salad.metaschema

Module Contents

Classes

LoadingOptions	
Saveable	Mark classes than have a save() and fromDoc() function.
Documented	Mark classes than have a save() and fromDoc() function.
RecordField	A field of a record.
RecordSchema	Mark classes than have a save() and fromDoc() function.
EnumSchema	Define an enumerated type.
ArraySchema	Mark classes than have a save() and fromDoc() function.
MapSchema	Mark classes than have a save() and fromDoc() function.
UnionSchema	Mark classes than have a save() and fromDoc() function.
JsonldPredicate	Attached to a record field to define how the parent record field is handled for
SpecializeDef	Mark classes than have a save() and fromDoc() function.
NamedType	Mark classes than have a save() and fromDoc() function.
DocType	Mark classes than have a save() and fromDoc() function.
SchemaDefinedType	Abstract base for schema-defined types.
SaladRecordField	A field of a record.
SaladRecordSchema	Mark classes than have a save() and fromDoc() function.
SaladEnumSchema	Define an enumerated type.
SaladMapSchema	Define a map type.
SaladUnionSchema	Define a union type.
Documentation	A documentation section. This type exists to facilitate self-documenting

Functions

load_field(val, fieldtype, baseuri, loadingOptions[, lc])	Load field.
extract_type(val_type)	Take a type of value, and extracts the value as a string.
convert_typing(val_type)	Normalize type names to schema-salad types.
parse_errors(error_message)	Parse error messages from several loaders into one error message.
save(val[, top, base_url, relative_uris])	
save_with_metadata(val, valLoadingOpts[, top, ...])	Save and set $namespaces, $schemas, $base and any other metadata fields at the top level.
expand_url(url, base_url, loadingOptions[, scoped_id, ...])	
file_uri(path[, split_frag])	Transform a file path into a URL with file scheme.
prefix_url(url, namespaces)	Expand short forms into full URLs using the given namespace dictionary.
save_relative_uri(uri, base_url, scoped_id, ref_scope, ...)	Convert any URI to a relative one, obeying the scoping rules.
shortname(inputid)	Compute the shortname of a fully qualified identifier.
parser_info()	
load_document(doc[, baseuri, loadingOptions])	
load_document_with_metadata(doc[, baseuri, ...])	
load_document_by_string(string, uri[, loadingOptions])	
load_document_by_yaml(yaml, uri[, loadingOptions])	Shortcut to load via a YAML object.

Attributes

IdxType
save_type
strtype
inttype
floattype
booltype
None_type
Any_type

continues on next page

Table 2 – continued from previous page

PrimitiveTypeLoader	Names of salad data types (based on Avro schema declarations).
AnyLoader	The **Any** type validates for any non-null value.
RecordFieldLoader	
RecordSchemaLoader	
EnumSchemaLoader	
ArraySchemaLoader	
MapSchemaLoader	
UnionSchemaLoader	
JsonldPredicateLoader	
SpecializeDefLoader	
SaladRecordFieldLoader	
SaladRecordSchemaLoader	
SaladEnumSchemaLoader	
SaladMapSchemaLoader	
SaladUnionSchemaLoader	
DocumentationLoader	
array_of_strtype	
union_of_None_type_or_strtype_or_array_of_st	
uri_strtype_True_False_None_None	
union_of_PrimitiveTypeLoader_or_RecordSchema	*der_or_MapSche*
array_of_union_of_PrimitiveTypeLoader_or_Rec	*SchemaLoader_o*
union_of_PrimitiveTypeLoader_or_RecordSchema	*der_or_MapSche*
typedsl_union_of_PrimitiveTypeLoader_or_Recc	*chemaLoader_or*
array_of_RecordFieldLoader	
union_of_None_type_or_array_of_RecordFieldLc	
*idmap_fields_union_of_None_type_or_array_of_*	

continues on next page

Table 2 – continued from previous page

Record_nameLoader
typedsl_Record_nameLoader_2
union_of_None_type_or_strtype
uri_union_of_None_type_or_strtype_True_Fals
uri_array_of_strtype_True_False_None_None
Enum_nameLoader
typedsl_Enum_nameLoader_2
uri_union_of_PrimitiveTypeLoader_or_RecordS *aLoader_or_Map*
Array_nameLoader
typedsl_Array_nameLoader_2
Map_nameLoader
typedsl_Map_nameLoader_2
Union_nameLoader
typedsl_Union_nameLoader_2
union_of_None_type_or_booltype
union_of_None_type_or_inttype
uri_strtype_False_False_1_None
uri_union_of_None_type_or_strtype_False_Fals
*uri_union_of_None_type_or_strtype_or_array_*
union_of_None_type_or_strtype_or_JsonldPred
union_of_None_type_or_Any_type
array_of_SaladRecordFieldLoader
union_of_None_type_or_array_of_SaladRecordF
*idmap_fields_union_of_None_type_or_array_of_*
*uri_union_of_None_type_or_strtype_or_array_*
array_of_SpecializeDefLoader

continues on next page

Table 2 – continued from previous page

union_of_None_type_or_array_of_SpecializeDe	
idmap_specialize_union_of_None_type_or_arra	
Documentation_nameLoader	
typedsl_Documentation_nameLoader_2	
union_of_SaladRecordSchemaLoader_or_SaladEn	*adUnionSchemaL*
array_of_union_of_SaladRecordSchemaLoader_o	*er_or_SaladUni*
union_of_SaladRecordSchemaLoader_or_SaladEn	*adUnionSchemaL*

schema_salad.metaschema.**IdxType**

class schema_salad.metaschema.**LoadingOptions**(*fetcher=None*, *namespaces=None*, *schemas=None*, *fileuri=None*, *copyfrom=None*, *original_doc=None*, *addl_metadata=None*, *baseuri=None*, *idx=None*, *imports=None*, *includes=None*, *no_link_check=None*, *container=None*)

Parameters

- **fetcher** (*Optional[schema_salad.fetcher.Fetcher]*) –
- **namespaces** (*Optional[Dict[str, str]]*) –
- **schemas** (*Optional[List[str]]*) –
- **fileuri** (*Optional[str]*) –
- **copyfrom** (*Optional[LoadingOptions]*) –
- **original_doc** (*Optional[Any]*) –
- **addl_metadata** (*Optional[Dict[str, str]]*) –
- **baseuri** (*Optional[str]*) –
- **idx** (*Optional[IdxType]*) –
- **imports** (*Optional[List[str]]*) –
- **includes** (*Optional[List[str]]*) –
- **no_link_check** (*Optional[bool]*) –
- **container** (*Optional[str]*) –

property graph: rdflib.Graph

Generate a merged rdflib.Graph from all entries in self.schemas.

Return type
rdflib.Graph

idx: IdxType

fileuri: str | None

baseuri: str

namespaces: MutableMapping[str, str]

schemas: MutableSequence[str]

original_doc: Any | None

addl_metadata: MutableMapping[str, Any]

fetcher: *schema_salad.fetcher.Fetcher*

vocab: Dict[str, str]

rvocab: Dict[str, str]

cache: schema_salad.utils.CacheType

imports: List[str]

includes: List[str]

no_link_check: bool | None

container: str | None

class schema_salad.metaschema.**Saveable**

Bases: abc.ABC

Mark classes than have a save() and fromDoc() function.

abstract classmethod fromDoc(*_doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **_doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type

Saveable

abstract save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type

Dict[str, Any]

schema_salad.metaschema.**load_field**(*val*, *fieldtype*, *baseuri*, *loadingOptions*, *lc=None*)

Load field.

Parameters

- **val** (*Union[str, Dict[str, str]]*) –
- **fieldtype** (*_Loader*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **lc** (*Optional[List[Any]]*) –

Return type
Any

schema_salad.metaschema.**save_type**

schema_salad.metaschema.**extract_type**(*val_type*)

Take a type of value, and extracts the value as a string.

Parameters
val_type (*Type[Any]*) –

Return type
str

schema_salad.metaschema.**convert_typing**(*val_type*)

Normalize type names to schema-salad types.

Parameters
val_type (*str*) –

Return type
str

schema_salad.metaschema.**parse_errors**(*error_message*)

Parse error messages from several loaders into one error message.

Parameters
error_message (*str*) –

Return type
Tuple[str, str, str]

schema_salad.metaschema.**save**(*val*, *top=True*, *base_url=''*, *relative_uris=True*)

Parameters

- **val** (*Any*) –
- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
save_type

schema_salad.metaschema.**save_with_metadata**(*val*, *valLoadingOpts*, *top=True*, *base_url=''*, *relative_uris=True*)

Save and set $namespaces, $schemas, $base and any other metadata fields at the top level.

Parameters

- **val** (*Any*) –
- **valLoadingOpts** (LoadingOptions) –
- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
save_type

schema_salad.metaschema.**expand_url**(*url*, *base_url*, *loadingOptions*, *scoped_id=False*, *vocab_term=False*, *scoped_ref=None*)

Parameters

- **url** (*str*) –
- **base_url** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **scoped_ref** (*Optional[int]*) –

Return type
str

schema_salad.metaschema.**file_uri**(*path*, *split_frag=False*)

Transform a file path into a URL with file scheme.

Parameters

- **path** (*str*) –
- **split_frag** (*bool*) –

Return type
str

schema_salad.metaschema.**prefix_url**(*url*, *namespaces*)

Expand short forms into full URLs using the given namespace dictionary.

Parameters

- **url** (*str*) –
- **namespaces** (*Dict[str, str]*) –

Return type
str

schema_salad.metaschema.**save_relative_uri**(*uri*, *base_url*, *scoped_id*, *ref_scope*, *relative_uris*)

Convert any URI to a relative one, obeying the scoping rules.

Parameters

- **uri** (*Any*) –
- **base_url** (*str*) –
- **scoped_id** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **relative_uris** (*bool*) –

Return type
Any

schema_salad.metaschema.**shortname**(*inputid*)

Compute the shortname of a fully qualified identifier.

See https://w3id.org/cwl/v1.2/SchemaSalad.html#Short_names.

Parameters
inputid (*str*) –

Return type
str

schema_salad.metaschema.**parser_info**()

Return type
str

class schema_salad.metaschema.**Documented**

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

class schema_salad.metaschema.**RecordField**(*name*, *type_*, *doc=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Documented*

A field of a record.

Parameters

- **name** (*Any*) –
- **type_** (*Any*) –
- **doc** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
RecordField

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**RecordSchema**(*type_*, *fields=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

Parameters

- **type_** (*Any*) –
- **fields** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
RecordSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**EnumSchema**(*symbols*, *type_*, *name=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Define an enumerated type.

Parameters

- **symbols** (*Any*) –
- **type_** (*Any*) –
- **name** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
EnumSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**ArraySchema**(*items*, *type_*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

Parameters

- **items** (*Any*) –
- **type_** (*Any*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
ArraySchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**MapSchema**(*type_*, *values*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

Parameters

- **type_** (*Any*) –
- **values** (*Any*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
MapSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**UnionSchema**(*names*, *type_*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

Parameters

- **names** (*Any*) –
- **type_** (*Any*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
UnionSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**JsonldPredicate**(*_id=None*, *_type=None*, *_container=None*, *identity=None*, *noLinkCheck=None*, *mapSubject=None*, *mapPredicate=None*, *refScope=None*, *typeDSL=None*, *secondaryFilesDSL=None*, *subscope=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Attached to a record field to define how the parent record field is handled for URI resolution and JSON-LD context generation.

Parameters

- **_id** (*Optional[Any]*) –
- **_type** (*Optional[Any]*) –
- **_container** (*Optional[Any]*) –

- **identity** (*Optional[Any]*) –
- **noLinkCheck** (*Optional[Any]*) –
- **mapSubject** (*Optional[Any]*) –
- **mapPredicate** (*Optional[Any]*) –
- **refScope** (*Optional[Any]*) –
- **typeDSL** (*Optional[Any]*) –
- **secondaryFilesDSL** (*Optional[Any]*) –
- **subscope** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (*LoadingOptions*) –
- **docRoot** (*Optional[str]*) –

Return type
JsonldPredicate

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**SpecializeDef**(*specializeFrom*, *specializeTo*, *extension_fields=None*, *loadingOptions=None*)

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

Parameters

- **specializeFrom** (*Any*) –
- **specializeTo** (*Any*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (*LoadingOptions*) –
- **docRoot** (*Optional[str]*) –

Return type
SpecializeDef

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**NamedType**

Bases: *Saveable*

Mark classes than have a save() and fromDoc() function.

class schema_salad.metaschema.**DocType**

Bases: *Documented*

Mark classes than have a save() and fromDoc() function.

class schema_salad.metaschema.**SchemaDefinedType**

Bases: *DocType*

Abstract base for schema-defined types.

class schema_salad.metaschema.**SaladRecordField**(*name*, *type_*, *doc=None*, *jsonldPredicate=None*, *default=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *RecordField*

A field of a record.

Parameters

- **name** (*Any*) –
- **type_** (*Any*) –
- **doc** (*Optional[Any]*) –
- **jsonldPredicate** (*Optional[Any]*) –
- **default** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
SaladRecordField

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**SaladRecordSchema**(*name*, *type_*, *inVocab=None*, *fields=None*, *doc=None*, *docParent=None*, *docChild=None*, *docAfter=None*, *jsonldPredicate=None*, *documentRoot=None*, *abstract=None*, *extends=None*, *specialize=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *NamedType*, *RecordSchema*, *SchemaDefinedType*

Mark classes than have a save() and fromDoc() function.

Parameters

- **name** (*Any*) –
- **type_** (*Any*) –
- **inVocab** (*Optional[Any]*) –
- **fields** (*Optional[Any]*) –
- **doc** (*Optional[Any]*) –
- **docParent** (*Optional[Any]*) –
- **docChild** (*Optional[Any]*) –
- **docAfter** (*Optional[Any]*) –
- **jsonldPredicate** (*Optional[Any]*) –
- **documentRoot** (*Optional[Any]*) –
- **abstract** (*Optional[Any]*) –
- **extends** (*Optional[Any]*) –
- **specialize** (*Optional[Any]*) –

- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (*LoadingOptions*) –
- **docRoot** (*Optional[str]*) –

Return type
SaladRecordSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**SaladEnumSchema**(*symbols*, *type_*, *name=None*, *inVocab=None*, *doc=None*, *docParent=None*, *docChild=None*, *docAfter=None*, *jsonldPredicate=None*, *documentRoot=None*, *extends=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *NamedType*, *EnumSchema*, *SchemaDefinedType*

Define an enumerated type.

Parameters

- **symbols** (*Any*) –

- **type_** (*Any*) –
- **name** (*Optional[Any]*) –
- **inVocab** (*Optional[Any]*) –
- **doc** (*Optional[Any]*) –
- **docParent** (*Optional[Any]*) –
- **docChild** (*Optional[Any]*) –
- **docAfter** (*Optional[Any]*) –
- **jsonldPredicate** (*Optional[Any]*) –
- **documentRoot** (*Optional[Any]*) –
- **extends** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (*LoadingOptions*) –
- **docRoot** (*Optional[str]*) –

Return type
SaladEnumSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**SaladMapSchema**(*name*, *type_*, *values*, *inVocab=None*, *doc=None*, *docParent=None*, *docChild=None*, *docAfter=None*, *jsonldPredicate=None*, *documentRoot=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *NamedType*, *MapSchema*, *SchemaDefinedType*

Define a map type.

Parameters

- **name** (*Any*) –
- **type_** (*Any*) –
- **values** (*Any*) –
- **inVocab** (*Optional[Any]*) –
- **doc** (*Optional[Any]*) –
- **docParent** (*Optional[Any]*) –
- **docChild** (*Optional[Any]*) –
- **docAfter** (*Optional[Any]*) –
- **jsonldPredicate** (*Optional[Any]*) –
- **documentRoot** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod **fromDoc**(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (*LoadingOptions*) –

- **docRoot** (*Optional[str]*) –

Return type
SaladMapSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**SaladUnionSchema**(*name*, *names*, *type_*, *inVocab=None*, *doc=None*, *docParent=None*, *docChild=None*, *docAfter=None*, *documentRoot=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *NamedType*, *UnionSchema*, *DocType*

Define a union type.

Parameters

- **name** (*Any*) –
- **names** (*Any*) –
- **type_** (*Any*) –
- **inVocab** (*Optional[Any]*) –
- **doc** (*Optional[Any]*) –
- **docParent** (*Optional[Any]*) –
- **docChild** (*Optional[Any]*) –
- **docAfter** (*Optional[Any]*) –
- **documentRoot** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
SaladUnionSchema

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

class schema_salad.metaschema.**Documentation**(*name*, *type_*, *inVocab=None*, *doc=None*, *docParent=None*, *docChild=None*, *docAfter=None*, *extension_fields=None*, *loadingOptions=None*)

Bases: *NamedType*, *DocType*

A documentation section. This type exists to facilitate self-documenting schemas but has no role in formal validation.

Parameters

- **name** (*Any*) –
- **type_** (*Any*) –
- **inVocab** (*Optional[Any]*) –
- **doc** (*Optional[Any]*) –
- **docParent** (*Optional[Any]*) –
- **docChild** (*Optional[Any]*) –
- **docAfter** (*Optional[Any]*) –
- **extension_fields** (*Optional[Dict[str, Any]]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

attrs

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__hash__()

Return hash(self).

Return type
int

classmethod fromDoc(*doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **docRoot** (*Optional[str]*) –

Return type
Documentation

save(*top=False*, *base_url=''*, *relative_uris=True*)

Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

schema_salad.metaschema.**strtype**

schema_salad.metaschema.**inttype**

schema_salad.metaschema.**floattype**

schema_salad.metaschema.**booltype**

schema_salad.metaschema.**None_type**

schema_salad.metaschema.**Any_type**

schema_salad.metaschema.**PrimitiveTypeLoader**

Names of salad data types (based on Avro schema declarations).

Refer to the [Avro schema declaration documentation](https://avro.apache.org/docs/current/spec.html#schemas) for detailed information.

null: no value boolean: a binary value int: 32-bit signed integer long: 64-bit signed integer float: single precision (32-bit) IEEE 754 floating-point number double: double precision (64-bit) IEEE 754 floating-point number string: Unicode character sequence

schema_salad.metaschema.**AnyLoader**

The **Any** type validates for any non-null value.

schema_salad.metaschema.**RecordFieldLoader**

schema_salad.metaschema.**RecordSchemaLoader**

schema_salad.metaschema.**EnumSchemaLoader**

schema_salad.metaschema.**ArraySchemaLoader**

schema_salad.metaschema.**MapSchemaLoader**

schema_salad.metaschema.**UnionSchemaLoader**

schema_salad.metaschema.**JsonldPredicateLoader**

schema_salad.metaschema.**SpecializeDefLoader**

schema_salad.metaschema.**SaladRecordFieldLoader**

schema_salad.metaschema.**SaladRecordSchemaLoader**

schema_salad.metaschema.**SaladEnumSchemaLoader**

schema_salad.metaschema.**SaladMapSchemaLoader**

schema_salad.metaschema.**SaladUnionSchemaLoader**

schema_salad.metaschema.**DocumentationLoader**

schema_salad.metaschema.**array_of_strtype**

schema_salad.metaschema.**union_of_None_type_or_strtype_or_array_of_strtype**

schema_salad.metaschema.**uri_strtype_True_False_None_None**

schema_salad.metaschema.
union_of_PrimitiveTypeLoader_or_RecordSchemaLoader_or_EnumSchemaLoader_or_ArraySchemaLoader_or_MapSchema

schema_salad.metaschema.
array_of_union_of_PrimitiveTypeLoader_or_RecordSchemaLoader_or_EnumSchemaLoader_or_ArraySchemaLoader_or

schema_salad.metaschema.
union_of_PrimitiveTypeLoader_or_RecordSchemaLoader_or_EnumSchemaLoader_or_ArraySchemaLoader_or_MapSchema

schema_salad.metaschema.
typedsl_union_of_PrimitiveTypeLoader_or_RecordSchemaLoader_or_EnumSchemaLoader_or_ArraySchemaLoader_or_

schema_salad.metaschema.**array_of_RecordFieldLoader**

schema_salad.metaschema.**union_of_None_type_or_array_of_RecordFieldLoader**

schema_salad.metaschema.**idmap_fields_union_of_None_type_or_array_of_RecordFieldLoader**

schema_salad.metaschema.**Record_nameLoader**

schema_salad.metaschema.**typedsl_Record_nameLoader_2**

schema_salad.metaschema.**union_of_None_type_or_strtype**

schema_salad.metaschema.**uri_union_of_None_type_or_strtype_True_False_None_None**

schema_salad.metaschema.**uri_array_of_strtype_True_False_None_None**

schema_salad.metaschema.**Enum_nameLoader**

schema_salad.metaschema.**typedsl_Enum_nameLoader_2**

schema_salad.metaschema.
uri_union_of_PrimitiveTypeLoader_or_RecordSchemaLoader_or_EnumSchemaLoader_or_ArraySchemaLoader_or_MapS

schema_salad.metaschema.**Array_nameLoader**

schema_salad.metaschema.**typedsl_Array_nameLoader_2**

schema_salad.metaschema.**Map_nameLoader**

schema_salad.metaschema.**typedsl_Map_nameLoader_2**

schema_salad.metaschema.**Union_nameLoader**

schema_salad.metaschema.**typedsl_Union_nameLoader_2**

schema_salad.metaschema.**union_of_None_type_or_booltype**

schema_salad.metaschema.**union_of_None_type_or_inttype**

schema_salad.metaschema.**uri_strtype_False_False_1_None**

schema_salad.metaschema.**uri_union_of_None_type_or_strtype_False_False_None_None**

schema_salad.metaschema.
uri_union_of_None_type_or_strtype_or_array_of_strtype_False_False_None_None

schema_salad.metaschema.**union_of_None_type_or_strtype_or_JsonldPredicateLoader**

schema_salad.metaschema.**union_of_None_type_or_Any_type**

schema_salad.metaschema.**array_of_SaladRecordFieldLoader**

schema_salad.metaschema.**union_of_None_type_or_array_of_SaladRecordFieldLoader**

schema_salad.metaschema.
idmap_fields_union_of_None_type_or_array_of_SaladRecordFieldLoader

schema_salad.metaschema.
uri_union_of_None_type_or_strtype_or_array_of_strtype_False_False_1_None

schema_salad.metaschema.**array_of_SpecializeDefLoader**

schema_salad.metaschema.**union_of_None_type_or_array_of_SpecializeDefLoader**

schema_salad.metaschema.
idmap_specialize_union_of_None_type_or_array_of_SpecializeDefLoader

schema_salad.metaschema.**Documentation_nameLoader**

schema_salad.metaschema.**typedsl_Documentation_nameLoader_2**

schema_salad.metaschema.
union_of_SaladRecordSchemaLoader_or_SaladEnumSchemaLoader_or_SaladMapSchemaLoader_or_SaladUnionSchemaLo

schema_salad.metaschema.
array_of_union_of_SaladRecordSchemaLoader_or_SaladEnumSchemaLoader_or_SaladMapSchemaLoader_or_SaladUnio

schema_salad.metaschema.
union_of_SaladRecordSchemaLoader_or_SaladEnumSchemaLoader_or_SaladMapSchemaLoader_or_SaladUnionSchemaLo

schema_salad.metaschema.**load_document**(*doc*, *baseuri=None*, *loadingOptions=None*)

Parameters

- **doc** (*Any*) –
- **baseuri** (*Optional[str]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

Return type
Any

schema_salad.metaschema.**load_document_with_metadata**(*doc*, *baseuri=None*, *loadingOptions=None*, *addl_metadata_fields=None*)

Parameters

- **doc** (*Any*) –
- **baseuri** (*Optional[str]*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –
- **addl_metadata_fields** (*Optional[MutableSequence[str]]*) –

Return type
Any

schema_salad.metaschema.**load_document_by_string**(*string*, *uri*, *loadingOptions=None*)

Parameters

- **string** (*Any*) –
- **uri** (*str*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

Return type
Any

schema_salad.metaschema.**load_document_by_yaml**(*yaml*, *uri*, *loadingOptions=None*)

Shortcut to load via a YAML object. yaml: must be from ruamel.yaml.main.YAML.load with preserve_quotes=True

Parameters

- **yaml** (*Any*) –
- **uri** (*str*) –
- **loadingOptions** (*Optional[LoadingOptions]*) –

Return type
Any

schema_salad.python_codegen

Python code generator for a given schema salad definition.

Module Contents

Classes

PythonCodeGen	Generation of Python code for a given Schema Salad definition.

Functions

fmt(text, indent)	Use black to format this snippet.

Attributes

black
prims

schema_salad.python_codegen.**black**

schema_salad.python_codegen.**prims**

schema_salad.python_codegen.**fmt**(*text*, *indent*)

Use black to format this snippet.

:param indent the indent level for the current context

Parameters

- **text** (*str*) –
- **indent** (*int*) –

Return type
str

class schema_salad.python_codegen.**PythonCodeGen**(*out*, *copyright*, *parser_info*, *salad_version*)

Bases: *schema_salad.codegen_base.CodeGenBase*

Generation of Python code for a given Schema Salad definition.

Parameters

- **out** (*IO[str]*) –
- **copyright** (*Optional[str]*) –
- **parser_info** (*str*) –
- **salad_version** (*str*) –

static **safe_name**(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

prologue()

Trigger to generate the prolouge code.

Return type
None

begin_class(*classname*, *extends*, *doc*, *abstract*, *field_names*, *idfield*, *optional_fields*)

Produce the header for the given class.

Parameters

- **classname** (*str*) –
- **extends** (*MutableSequence[str]*) –
- **doc** (*str*) –
- **abstract** (*bool*) –
- **field_names** (*MutableSequence[str]*) –
- **idfield** (*str*) –
- **optional_fields** (*Set[str]*) –

Return type
None

end_class(*classname*, *field_names*)

Signal that we are done with this class.

Parameters

- **classname** (*str*) –
- **field_names** (*List[str]*) –

Return type
None

type_loader(*type_declaration*, *container=None*, *no_link_check=None*)

Parse the given type declaration and declare its components.

Parameters

- **type_declaration** (*Union[List[Any], Dict[str, Any], str]*) –
- **container** (*Optional[str]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

declare_id_field(*name*, *fieldtype*, *doc*, *optional*)

Output the code to handle the given ID field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –

Return type
None

declare_field(*name*, *fieldtype*, *doc*, *optional*, *subscope*)

Output the code to load the given field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –
- **subscope** (*Optional[str]*) –

Return type
None

uri_loader(*inner*, *scoped_id*, *vocab_term*, *ref_scope*, *no_link_check=None*)

Construct the TypeDef for the given URI loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

idmap_loader(*field*, *inner*, *map_subject*, *map_predicate*)

Construct the TypeDef for the given mapped ID loader.

Parameters

- **field** (*str*) –
- **inner** (schema_salad.codegen_base.TypeDef) –
- **map_subject** (*str*) –
- **map_predicate** (*Optional[str]*) –

Return type
schema_salad.codegen_base.TypeDef

typedsl_loader(*inner*, *ref_scope*)

Construct the TypeDef for the given DSL loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **ref_scope** (*Optional[int]*) –

Return type
schema_salad.codegen_base.TypeDef

secondaryfilesdsl_loader(*inner*)

Construct the TypeDef for secondary files.

Parameters
inner (schema_salad.codegen_base.TypeDef) –

Return type
schema_salad.codegen_base.TypeDef

epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (schema_salad.codegen_base.TypeDef) –

Return type
None

schema_salad.python_codegen_support

Template code used by python_codegen.py.

Module Contents

Classes

LoadingOptions	
Saveable	Mark classes than have a save() and fromDoc() function.

Functions

load_field(val, fieldtype, baseuri, loadingOptions[, lc])	Load field.
extract_type(val_type)	Take a type of value, and extracts the value as a string.
convert_typing(val_type)	Normalize type names to schema-salad types.
parse_errors(error_message)	Parse error messages from several loaders into one error message.
save(val[, top, base_url, relative_uris])	
save_with_metadata(val, valLoadingOpts[, top, ...])	Save and set $namespaces, $schemas, $base and any other metadata fields at the top level.
expand_url(url, base_url, loadingOptions[, scoped_id, ...])	
file_uri(path[, split_frag])	Transform a file path into a URL with file scheme.
prefix_url(url, namespaces)	Expand short forms into full URLs using the given namespace dictionary.
save_relative_uri(uri, base_url, scoped_id, ref_scope, ...)	Convert any URI to a relative one, obeying the scoping rules.
shortname(inputid)	Compute the shortname of a fully qualified identifier.

Attributes

IdxType
save_type

schema_salad.python_codegen_support.**IdxType**

class schema_salad.python_codegen_support.**LoadingOptions**(*fetcher=None*, *namespaces=None*, *schemas=None*, *fileuri=None*, *copyfrom=None*, *original_doc=None*, *addl_metadata=None*, *baseuri=None*, *idx=None*, *imports=None*, *includes=None*, *no_link_check=None*, *container=None*)

Parameters

- **fetcher** (*Optional[schema_salad.fetcher.Fetcher]*) –
- **namespaces** (*Optional[Dict[str, str]]*) –
- **schemas** (*Optional[List[str]]*) –
- **fileuri** (*Optional[str]*) –
- **copyfrom** (*Optional[LoadingOptions]*) –
- **original_doc** (*Optional[Any]*) –
- **addl_metadata** (*Optional[Dict[str, str]]*) –

- **baseuri** (*Optional[str]*) –
- **idx** (*Optional[IdxType]*) –
- **imports** (*Optional[List[str]]*) –
- **includes** (*Optional[List[str]]*) –
- **no_link_check** (*Optional[bool]*) –
- **container** (*Optional[str]*) –

property graph: rdflib.Graph

Generate a merged rdflib.Graph from all entries in self.schemas.

Return type
rdflib.Graph

idx: IdxType

fileuri: str | None

baseuri: str

namespaces: MutableMapping[str, str]

schemas: MutableSequence[str]

original_doc: Any | None

addl_metadata: MutableMapping[str, Any]

fetcher: *schema_salad.fetcher.Fetcher*

vocab: Dict[str, str]

rvocab: Dict[str, str]

cache: schema_salad.utils.CacheType

imports: List[str]

includes: List[str]

no_link_check: bool | None

container: str | None

class schema_salad.python_codegen_support.**Saveable**

Bases: abc.ABC

Mark classes than have a save() and fromDoc() function.

abstract classmethod fromDoc(*_doc*, *baseuri*, *loadingOptions*, *docRoot=None*)

Construct this object from the result of yaml.load().

Parameters

- **_doc** (*Any*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –

- **docRoot** (*Optional[str]*) –

Return type
Saveable

abstract save(*top=False*, *base_url=''*, *relative_uris=True*)
Convert this object to a JSON/YAML friendly dictionary.

Parameters

- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
Dict[str, Any]

schema_salad.python_codegen_support.**load_field**(*val*, *fieldtype*, *baseuri*, *loadingOptions*, *lc=None*)
Load field.

Parameters

- **val** (*Union[str, Dict[str, str]]*) –
- **fieldtype** (*_Loader*) –
- **baseuri** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **lc** (*Optional[List[Any]]*) –

Return type
Any

schema_salad.python_codegen_support.**save_type**

schema_salad.python_codegen_support.**extract_type**(*val_type*)
Take a type of value, and extracts the value as a string.

Parameters
val_type (*Type[Any]*) –

Return type
str

schema_salad.python_codegen_support.**convert_typing**(*val_type*)
Normalize type names to schema-salad types.

Parameters
val_type (*str*) –

Return type
str

schema_salad.python_codegen_support.**parse_errors**(*error_message*)
Parse error messages from several loaders into one error message.

Parameters
error_message (*str*) –

Return type
Tuple[str, str, str]

schema_salad.python_codegen_support.**save**(*val*, *top=True*, *base_url=''*, *relative_uris=True*)

Parameters

- **val** (*Any*) –
- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
save_type

schema_salad.python_codegen_support.**save_with_metadata**(*val*, *valLoadingOpts*, *top=True*, *base_url=''*, *relative_uris=True*)

Save and set $namespaces, $schemas, $base and any other metadata fields at the top level.

Parameters

- **val** (*Any*) –
- **valLoadingOpts** (LoadingOptions) –
- **top** (*bool*) –
- **base_url** (*str*) –
- **relative_uris** (*bool*) –

Return type
save_type

schema_salad.python_codegen_support.**expand_url**(*url*, *base_url*, *loadingOptions*, *scoped_id=False*, *vocab_term=False*, *scoped_ref=None*)

Parameters

- **url** (*str*) –
- **base_url** (*str*) –
- **loadingOptions** (LoadingOptions) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **scoped_ref** (*Optional[int]*) –

Return type
str

schema_salad.python_codegen_support.**file_uri**(*path*, *split_frag=False*)

Transform a file path into a URL with file scheme.

Parameters

- **path** (*str*) –
- **split_frag** (*bool*) –

Return type
str

schema_salad.python_codegen_support.**prefix_url**(*url*, *namespaces*)

Expand short forms into full URLs using the given namespace dictionary.

Parameters

- **url** (*str*) –
- **namespaces** (*Dict[str, str]*) –

Return type
str

schema_salad.python_codegen_support.**save_relative_uri**(*uri*, *base_url*, *scoped_id*, *ref_scope*, *relative_uris*)

Convert any URI to a relative one, obeying the scoping rules.

Parameters

- **uri** (*Any*) –
- **base_url** (*str*) –
- **scoped_id** (*bool*) –
- **ref_scope** (*Optional[int]*) –
- **relative_uris** (*bool*) –

Return type
Any

schema_salad.python_codegen_support.**shortname**(*inputid*)

Compute the shortname of a fully qualified identifier.

See https://w3id.org/cwl/v1.2/SchemaSalad.html#Short_names.

Parameters
inputid (*str*) –

Return type
str

schema_salad.ref_resolver

Module Contents

Classes

NormDict	A Dict where all keys are normalized using the provided function.
Loader	

Functions

file_uri(path[, split_frag])	
uri_file_path(url)	
to_validation_exception(e)	Convert ruamel.yaml exception to our type.
SubLoader(loader)	

schema_salad.ref_resolver.**file_uri**(*path*, *split_frag=False*)

Parameters

- **path** (*str*) –
- **split_frag** (*bool*) –

Return type
str

schema_salad.ref_resolver.**uri_file_path**(*url*)

Parameters
url (*str*) –

Return type
str

schema_salad.ref_resolver.**to_validation_exception**(*e*)

Convert ruamel.yaml exception to our type.

Parameters
e (*ruamel.yaml.error.MarkedYAMLError*) –

Return type
schema_salad.exceptions.ValidationException

class schema_salad.ref_resolver.**NormDict**(*normalize=str*)

Bases: Dict[str, Union[ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq, str, None]]

A Dict where all keys are normalized using the provided function.

Parameters
normalize (*Callable[[str], str]*) –

__eq__(*other*)

Return self==value.

Parameters
other (*Any*) –

Return type
bool

__getitem__(*key*)

x.__getitem__(y) <==> x[y]

Parameters
key (*Any*) –

Return type
Any

__setitem__(*key*, *value*)

Set self[key] to value.

Parameters

- **key** (*Any*) –
- **value** (*Any*) –

Return type
Any

__delitem__(*key*)

Delete self[key].

Parameters
key (*Any*) –

Return type
Any

__contains__(*key*)

True if the dictionary has the specified key, else False.

Parameters
key (*Any*) –

Return type
bool

__del__()

Return type
None

schema_salad.ref_resolver.**SubLoader**(*loader*)

Parameters
loader (Loader) –

Return type
Loader

class schema_salad.ref_resolver.**Loader**(*ctx*, *schemagraph=None*, *foreign_properties=None*, *idx=None*, *cache=None*, *session=None*, *fetcher_constructor=None*, *skip_schemas=None*, *url_fields=None*, *allow_attachments=None*, *doc_cache=True*, *salad_version=None*)

Parameters

- **ctx** (*schema_salad.utils.ContextType*) –
- **schemagraph** (*Optional[rdflib.graph.Graph]*) –
- **foreign_properties** (*Optional[Set[str]]*) –

- **idx** (*Optional[schema_salad.utils.IdxType]*) –
- **cache** (*Optional[schema_salad.utils.CacheType]*) –
- **session** (*Optional[requests.sessions.Session]*) –
- **fetcher_constructor** (*Optional[schema_salad.utils.FetcherCallableType]*) –
- **skip_schemas** (*Optional[bool]*) –
- **url_fields** (*Optional[Set[str]]*) –
- **allow_attachments** (*Optional[schema_salad.utils.AttachmentsType]*) –
- **doc_cache** (*Union[str, bool]*) –
- **salad_version** (*Optional[str]*) –

expand_url(*url*, *base_url*, *scoped_id=False*, *vocab_term=False*, *scoped_ref=None*)

Parameters

- **url** (*str*) –
- **base_url** (*str*) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –
- **scoped_ref** (*Optional[int]*) –

Return type
str

add_namespaces(*ns*)

Parameters
ns (*Dict[str, str]*) –

Return type
None

add_schemas(*ns*, *base_url*)

Parameters

- **ns** (*Union[List[str], str]*) –
- **base_url** (*str*) –

Return type
None

add_context(*newcontext*)

Parameters
newcontext (*schema_salad.utils.ContextType*) –

Return type
None

resolve_ref(*ref*, *base_url=None*, *checklinks=True*, *strict_foreign_properties=False*, *content_types=None*)

Parameters

- **ref** (*schema_salad.utils.ResolveType*) –

- **base_url** (*Optional[str]*) –
- **checklinks** (*bool*) –
- **strict_foreign_properties** (*bool*) –
- **content_types** (*Optional[List[str]]*) –

Return type
schema_salad.utils.ResolvedRefType

resolve_all(*document*, *base_url*, *file_base=None*, *checklinks=True*, *strict_foreign_properties=False*)

Parameters

- **document** (*schema_salad.utils.ResolveType*) –
- **base_url** (*str*) –
- **file_base** (*Optional[str]*) –
- **checklinks** (*bool*) –
- **strict_foreign_properties** (*bool*) –

Return type
schema_salad.utils.ResolvedRefType

fetch(*url*, *inject_ids=True*, *content_types=None*)

Parameters

- **url** (*str*) –
- **inject_ids** (*bool*) –
- **content_types** (*Optional[List[str]]*) –

Return type
schema_salad.utils.IdxResultType

validate_scoped(*field*, *link*, *docid*)

Parameters

- **field** (*str*) –
- **link** (*str*) –
- **docid** (*str*) –

Return type
str

validate_link(*field*, *link*, *docid*, *all_doc_ids*)

Parameters

- **field** (*str*) –
- **link** (*Union[str, ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap]*) –
- **docid** (*str*) –
- **all_doc_ids** (*Dict[str, str]*) –

Return type
Union[str, ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap]

getid(*d*)

Parameters
d (*Any*) –

Return type
Optional[str]

validate_links(*document*, *base_url*, *all_doc_ids*, *strict_foreign_properties=False*)

Parameters

- **document** (*schema_salad.utils.ResolveType*) –
- **base_url** (*str*) –
- **all_doc_ids** (*Dict[str, str]*) –
- **strict_foreign_properties** (*bool*) –

Return type
None

schema_salad.schema

Functions to process Schema Salad schemas.

Module Contents

Functions

get_metaschema()	Instantiate the metaschema.
add_namespaces(metadata, namespaces)	Collect the provided namespaces, checking for conflicts.
collect_namespaces(metadata)	Walk through the metadata object, collecting namespace declarations.
load_schema(schema_ref[, cache])	Load a schema that can be used to validate documents using load_and_validate.
load_and_validate(document_loader, avsc_names, ...[, ...])	Load a document and validate it with the provided schema.
validate_doc(schema_names, doc, loader, strict[, ...])	Validate a document using the provided schema.
get_anon_name(rec)	Calculate a reproducible name for anonymous types.
replace_type(items, spec, loader, found[, ...])	Go through and replace types in the 'spec' mapping.
avro_field_name(url)	Turn a URL into an Avro-safe name.
make_valid_avro(items, alltypes, found[, union, ...])	Convert our schema to be more avro like.
deepcopy_strip(item)	Make a deep copy of list and dict objects.
extend_and_specialize(items, loader)	Apply 'extend' and 'specialize' to fully materialize derived record types.
make_avro(i, loader[, metaschema_vocab])	
make_avro_schema(i, loader[, metaschema_vocab])	All in one convenience function.
make_avro_schema_from_avro(avro)	
shortname(inputid)	Return the last segment of the provided fragment or path.
print_inheritance(doc, stream)	Write a Grapviz inheritance graph for the supplied document.
print_fieldrefs(doc, loader, stream)	Write a GraphViz graph of the relationships between the fields.

Attributes

SALAD_FILES
saladp
cached_metaschema
schema_type
Avro

schema_salad.schema.**SALAD_FILES = ('metaschema.yml', 'metaschema_base.yml', 'salad.md', 'field_name.yml', 'import_include.md',...**

schema_salad.schema.**saladp = 'https://w3id.org/cwl/salad#'**

schema_salad.schema.**cached_metaschema: Tuple[*schema_salad.avro.schema.Names*, List[Dict[str, str]], *schema_salad.ref_resolver.Loader*] | None**

schema_salad.schema.**get_metaschema**()

Instantiate the metaschema.

Return type
Tuple[*schema_salad.avro.schema.Names*, List[Dict[str, str]], *schema_salad.ref_resolver.Loader*]

schema_salad.schema.**add_namespaces**(*metadata*, *namespaces*)

Collect the provided namespaces, checking for conflicts.

Parameters

- **metadata** (*Mapping[str, Any]*) –
- **namespaces** (*MutableMapping[str, str]*) –

Return type
None

schema_salad.schema.**collect_namespaces**(*metadata*)

Walk through the metadata object, collecting namespace declarations.

Parameters
metadata (*Mapping[str, Any]*) –

Return type
Dict[str, str]

schema_salad.schema.**schema_type**

schema_salad.schema.**load_schema**(*schema_ref*, *cache=None*)

Load a schema that can be used to validate documents using load_and_validate.

Returns
document_loader, avsc_names, schema_metadata, metaschema_loader

Parameters

- **schema_ref** (*schema_salad.utils.ResolveType*) –
- **cache** (*Optional[schema_salad.utils.CacheType]*) –

Return type
schema_type

schema_salad.schema.**load_and_validate**(*document_loader*, *avsc_names*, *document*, *strict*, *strict_foreign_properties=False*)

Load a document and validate it with the provided schema.

return data, metadata

Parameters

- **document_loader** (schema_salad.ref_resolver.Loader) –
- **avsc_names** (schema_salad.avro.schema.Names) –
- **document** (*Union[ruamel.yaml.comments.CommentedMap, str]*) –
- **strict** (*bool*) –
- **strict_foreign_properties** (*bool*) –

Return type
Tuple[Any, Dict[str, Any]]

schema_salad.schema.**validate_doc**(*schema_names*, *doc*, *loader*, *strict*, *strict_foreign_properties=False*)

Validate a document using the provided schema.

Parameters

- **schema_names** (schema_salad.avro.schema.Names) –
- **doc** (*schema_salad.utils.ResolveType*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **strict** (*bool*) –
- **strict_foreign_properties** (*bool*) –

Return type
None

schema_salad.schema.**get_anon_name**(*rec*)

Calculate a reproducible name for anonymous types.

Parameters
rec (*MutableMapping[str, Union[str, Dict[str, str], List[str]]]*) –

Return type
str

schema_salad.schema.**replace_type**(*items*, *spec*, *loader*, *found*, *find_embeds=True*, *deepen=True*)

Go through and replace types in the 'spec' mapping.

Parameters

- **items** (*Any*) –
- **spec** (*Dict[str, Any]*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **found** (*Set[str]*) –
- **find_embeds** (*bool*) –
- **deepen** (*bool*) –

Return type
Any

schema_salad.schema.**avro_field_name**(*url*)

Turn a URL into an Avro-safe name.

If the URL has no fragment, return this plain URL.

Extract either the last part of the URL fragment past the slash, otherwise the whole fragment.

Parameters
url (*str*) –

Return type
str

schema_salad.schema.**Avro**

schema_salad.schema.**make_valid_avro**(*items*, *alltypes*, *found*, *union=False*, *fielddef=False*, *vocab=None*)

Convert our schema to be more avro like.

Parameters

- **items** (*Avro*) –
- **alltypes** (*Dict[str, Dict[str, Any]]*) –
- **found** (*Set[str]*) –
- **union** (*bool*) –
- **fielddef** (*bool*) –
- **vocab** (*Optional[Dict[str, str]]*) –

Return type
Union[Avro, MutableMapping[str, str], str, List[Union[Any, MutableMapping[str, str], str]]]

schema_salad.schema.**deepcopy_strip**(*item*)

Make a deep copy of list and dict objects.

Intentionally do not copy attributes. This is to discard CommentedMap and CommentedSeq metadata which is very expensive with regular copy.deepcopy.

Parameters
item (*Any*) –

Return type
Any

schema_salad.schema.**extend_and_specialize**(*items*, *loader*)

Apply 'extend' and 'specialize' to fully materialize derived record types.

Parameters

- **items** (*List[Dict[str, Any]]*) –
- **loader** (schema_salad.ref_resolver.Loader) –

Return type
List[Dict[str, Any]]

schema_salad.schema.**make_avro**(*i*, *loader*, *metaschema_vocab=None*)

Parameters

- **i** (*List[Dict[str, Any]]*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **metaschema_vocab** (*Optional[Dict[str, str]]*) –

Return type
List[Any]

schema_salad.schema.**make_avro_schema**(*i*, *loader*, *metaschema_vocab=None*)

All in one convenience function.

Call make_avro() and make_avro_schema_from_avro() separately if you need the intermediate result for diagnostic output.

Parameters

- **i** (*List[Any]*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **metaschema_vocab** (*Optional[Dict[str, str]]*) –

Return type
schema_salad.avro.schema.Names

schema_salad.schema.**make_avro_schema_from_avro**(*avro*)

Parameters
avro (*List[Union[Avro, Dict[str, str], str]]*) –

Return type
schema_salad.avro.schema.Names

schema_salad.schema.**shortname**(*inputid*)

Return the last segment of the provided fragment or path.

Parameters
inputid (*str*) –

Return type
str

schema_salad.schema.**print_inheritance**(*doc*, *stream*)

Write a Grapviz inheritance graph for the supplied document.

Parameters

- **doc** (*List[Dict[str, Any]]*) –
- **stream** (*IO[Any]*) –

Return type
None

schema_salad.schema.**print_fieldrefs**(*doc*, *loader*, *stream*)

Write a GraphViz graph of the relationships between the fields.

Parameters

- **doc** (*List[Dict[str, Any]]*) –
- **loader** (schema_salad.ref_resolver.Loader) –
- **stream** (*IO[Any]*) –

Return type
None

schema_salad.sourceline

Module Contents

Classes

SourceLine

Functions

relname(source)	
add_lc_filename(r, source)	
reflow_all(text[, maxline])	
reflow(text, maxline[, shift])	
indent(v[, nolead, shift, bullet])	
bullets(textlist, bul)	
strip_duplicated_lineno(text)	Strip duplicated line numbers.
strip_dup_lineno(text[, maxline])	
cmap(d[, lc, fn])	

Attributes

lineno_re

schema_salad.sourceline.**lineno_re**

schema_salad.sourceline.**relname**(*source*)

Parameters
source (*str*) –

Return type
str

schema_salad.sourceline.**add_lc_filename**(*r*, *source*)

Parameters

- **r** (*ruamel.yaml.comments.CommentedBase*) –
- **source** (*str*) –

Return type
None

schema_salad.sourceline.**reflow_all**(*text*, *maxline=None*)

Parameters

- **text** (*str*) –
- **maxline** (*Optional[int]*) –

Return type
str

schema_salad.sourceline.**reflow**(*text*, *maxline*, *shift=''*)

Parameters

- **text** (*str*) –
- **maxline** (*int*) –
- **shift** (*Optional[str]*) –

Return type
str

schema_salad.sourceline.**indent**(*v*, *nolead=False*, *shift=' '*, *bullet=' '*)

Parameters

- **v** (*str*) –
- **nolead** (*bool*) –
- **shift** (*str*) –
- **bullet** (*str*) –

Return type
str

schema_salad.sourceline.**bullets**(*textlist*, *bul*)

Parameters

- **textlist** (*List[str]*) –
- **bul** (*str*) –

Return type
str

schema_salad.sourceline.**strip_duplicated_lineno**(*text*)

Strip duplicated line numbers.

Same as *strip_dup_lineno()* but without reflow.

Parameters
text (*str*) –

Return type
str

schema_salad.sourceline.**strip_dup_lineno**(*text*, *maxline=None*)

Parameters

- **text** (*str*) –
- **maxline** (*Optional[int]*) –

Return type
str

schema_salad.sourceline.**cmap**(*d*, *lc=None*, *fn=None*)

Parameters

- **d** (*Union[int, float, str, MutableMapping[str, Any], MutableSequence[Any], None]*) –
- **lc** (*Optional[List[int]]*) –
- **fn** (*Optional[str]*) –

Return type
Union[int, float, str, ruamel.yaml.comments.CommentedMap, ruamel.yaml.comments.CommentedSeq, None]

class schema_salad.sourceline.**SourceLine**(*item*, *key=None*, *raise_type=str*, *include_traceback=False*)

Parameters

- **item** (*Any*) –
- **key** (*Optional[Any]*) –
- **raise_type** (*Callable[[str], Any]*) –
- **include_traceback** (*bool*) –

__enter__()

Return type
SourceLine

__exit__(*exc_type*, *exc_value*, *tb*)

Parameters

- **exc_type** (*Any*) –
- **exc_value** (*Any*) –
- **tb** (*Any*) –

Return type
None

file()

Return type
Optional[str]

start()

Return type
Optional[Tuple[int, int]]

end()

Return type
Optional[Tuple[int, int]]

makeLead()

Return type
str

makeError(*msg*)

Parameters
msg (*str*) –

Return type
Any

schema_salad.typescript_codegen

TypeScript code generator for a given schema salad definition.

Module Contents

Classes

TypeScriptCodeGen	Generation of TypeScript code for a given Schema Salad definition.

Functions

doc_to_doc_string(doc[, indent_level])	Generate a documentation string from a schema salad doc field.

Attributes

prims

schema_salad.typescript_codegen.**doc_to_doc_string**(*doc*, *indent_level=0*)
Generate a documentation string from a schema salad doc field.

Parameters
- **doc** (*Optional[str]*) –
- **indent_level** (*int*) –

Return type
str

schema_salad.typescript_codegen.**prims**

class schema_salad.typescript_codegen.**TypeScriptCodeGen**(*base*, *examples*, *target*, *package*)
Bases: *schema_salad.codegen_base.CodeGenBase*

Generation of TypeScript code for a given Schema Salad definition.

Parameters

- **base** (*str*) –
- **examples** (*Optional[str]*) –
- **target** (*Optional[str]*) –
- **package** (*str*) –

prologue()

Trigger to generate the prolouge code.

Return type
None

static safe_name(*name*)

Generate a safe version of the given name.

Parameters
name (*str*) –

Return type
str

begin_class(*classname*, *extends*, *doc*, *abstract*, *field_names*, *idfield*, *optional_fields*)

Produce the header for the given class.

Parameters

- **classname** (*str*) –
- **extends** (*MutableSequence[str]*) –
- **doc** (*str*) –
- **abstract** (*bool*) –
- **field_names** (*MutableSequence[str]*) –
- **idfield** (*str*) –
- **optional_fields** (*Set[str]*) –

Return type
None

end_class(*classname*, *field_names*)

Signal that we are done with this class.

Parameters

- **classname** (*str*) –
- **field_names** (*List[str]*) –

Return type
None

type_loader(*type_declaration*, *container=None*, *no_link_check=None*)

Parse the given type declaration and declare its components.

Parameters

- **type_declaration** (*Union[List[Any], Dict[str, Any], str]*) –
- **container** (*Optional[str]*) –

- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

type_loader_enum(*type_declaration*)

Parameters
type_declaration (*Dict[str, Any]*) –

Return type
schema_salad.codegen_base.TypeDef

declare_field(*name*, *fieldtype*, *doc*, *optional*, *subscope*)

Output the code to load the given field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –
- **subscope** (*Optional[str]*) –

Return type
None

declare_id_field(*name*, *fieldtype*, *doc*, *optional*)

Output the code to handle the given ID field.

Parameters

- **name** (*str*) –
- **fieldtype** (schema_salad.codegen_base.TypeDef) –
- **doc** (*Optional[str]*) –
- **optional** (*bool*) –

Return type
None

to_typescript(*val*)

Convert a Python keyword to a TypeScript keyword.

Parameters
val (*Any*) –

Return type
Any

uri_loader(*inner*, *scoped_id*, *vocab_term*, *ref_scope*, *no_link_check=None*)

Construct the TypeDef for the given URI loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **scoped_id** (*bool*) –
- **vocab_term** (*bool*) –

- **ref_scope** (*Optional[int]*) –
- **no_link_check** (*Optional[bool]*) –

Return type
schema_salad.codegen_base.TypeDef

idmap_loader(*field*, *inner*, *map_subject*, *map_predicate*)

Construct the TypeDef for the given mapped ID loader.

Parameters

- **field** (*str*) –
- **inner** (schema_salad.codegen_base.TypeDef) –
- **map_subject** (*str*) –
- **map_predicate** (*Optional[str]*) –

Return type
schema_salad.codegen_base.TypeDef

typedsl_loader(*inner*, *ref_scope*)

Construct the TypeDef for the given DSL loader.

Parameters

- **inner** (schema_salad.codegen_base.TypeDef) –
- **ref_scope** (*Optional[int]*) –

Return type
schema_salad.codegen_base.TypeDef

epilogue(*root_loader*)

Trigger to generate the epilouge code.

Parameters
root_loader (schema_salad.codegen_base.TypeDef) –

Return type
None

secondaryfilesdsl_loader(*inner*)

Construct the TypeDef for secondary files.

Parameters
inner (schema_salad.codegen_base.TypeDef) –

Return type
schema_salad.codegen_base.TypeDef

schema_salad.utils

schema_salad.validate

Module Contents

Functions

validate(expected_schema, datum[, identifiers, ...])	
avro_shortname(name)	Produce an avro friendly short name.
avro_type_name(url)	Turn a URL into an Avro-safe name.
friendly(v)	Format an Avro schema into a pretty-printed representation.
vpformat(datum)	Truncate a pretty-printed representation of a Python object to 160 characters.
validate_ex(expected_schema, datum[, identifiers, ...])	Determine if a python datum is an instance of a schema.

Attributes

INT_MIN_VALUE
INT_MAX_VALUE
LONG_MIN_VALUE
LONG_MAX_VALUE
saladp
primitives

schema_salad.validate.**validate**(*expected_schema*, *datum*, *identifiers=None*, *strict=False*, *foreign_properties=None*, *vocab=None*)

Parameters

- **expected_schema** (schema_salad.avro.schema.Schema) –
- **datum** (*Any*) –
- **identifiers** (*Optional[List[str]]*) –
- **strict** (*bool*) –
- **foreign_properties** (*Optional[Set[str]]*) –
- **vocab** (*Optional[Mapping[str, str]]*) –

Return type
bool

schema_salad.validate.**INT_MIN_VALUE**

schema_salad.validate.**INT_MAX_VALUE**

schema_salad.validate.**LONG_MIN_VALUE**

schema_salad.validate.**LONG_MAX_VALUE**

schema_salad.validate.**avro_shortname**(*name*)

Produce an avro friendly short name.

Parameters
name (*str*) –

Return type
str

schema_salad.validate.**saladp = 'https://w3id.org/cwl/salad#'**

schema_salad.validate.**primitives**

schema_salad.validate.**avro_type_name**(*url*)

Turn a URL into an Avro-safe name.

If the URL has no fragment, return this plain URL.

Extract either the last part of the URL fragment past the slash, otherwise the whole fragment.

Parameters
url (*str*) –

Return type
str

schema_salad.validate.**friendly**(*v*)

Format an Avro schema into a pretty-printed representation.

Parameters
v (*Any*) –

Return type
Any

schema_salad.validate.**vpformat**(*datum*)

Truncate a pretty-printed representation of a Python object to 160 characters.

Parameters
datum (*Any*) –

Return type
str

schema_salad.validate.**validate_ex**(*expected_schema*, *datum*, *identifiers=None*, *strict=False*, *foreign_properties=None*, *raise_ex=True*, *strict_foreign_properties=False*, *logger=_logger*, *skip_foreign_properties=False*, *vocab=None*)

Determine if a python datum is an instance of a schema.

Parameters

- **expected_schema** (schema_salad.avro.schema.Schema) –
- **datum** (*Any*) –
- **identifiers** (*Optional[List[str]]*) –
- **strict** (*bool*) –
- **foreign_properties** (*Optional[Set[str]]*) –
- **raise_ex** (*bool*) –
- **strict_foreign_properties** (*bool*) –
- **logger** (*logging.Logger*) –
- **skip_foreign_properties** (*bool*) –
- **vocab** (*Optional[Mapping[str, str]]*) –

Return type
bool

Package Contents

schema_salad.**__author__** = 'peter.amstutz@curoverse.com'

6.4 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

H

J

L

M

N

O

P

Q

R

S

T

U

V

W