The SRS package: Reference documentation

srs

- get-all-items()
- get-class()
- get-class-namer-identifier()
- get-full-class()
- get-item()
- get-item-name-id()
- make-class()
- make-config()
- make-enum-field()
- make-field()
- make-item()
- make-origins()
- make-tag()
- show-items()
- show-template()
- tag-to-class-tree()
- validate-config()

Variables

- content-field

get-all-items

Returns all the items that belong to the given class given by the `tag`.

Parameters

```
get-all-items(
  items: dictionary,
  tag: array
) -> array
```

items `dictionary`

The item tree.

tag `array`

The tag

get-class

Returns the class matching the tag.

Parameters

```
get-class(
  config: dictionary,
  tag: array
) -> dictionary
```

config `dictionary`

Main config.

tag `array`

Tag of the class to find.

get-class-namer-identifier

Obtains the namer and identifier for a class, taking into account autos.

Parameters

```
get-class-namer-identifier(
  config: dictionary,
  tag,
  class: dictionary
) -> array
```

config `dictionary`

The full config

class `dictionary`

The class

get-full-class

This function merges all the fields of a class and subclasses identified by the `tag` into a big class which contains all the fields. The name of the resulting class is the name of the youngest child. This is a helper function.

Parameters

```
get-full-class(
  config: dictionary,
  tag: array
) -> dictionary
```

config `dictionary`

Guess what?! The configuration! lmao

tag `array`

Tag

get-item

Returns all the items that belong to the given class given by the `tag`.

Parameters

```
get-item(
  items: dictionary,
  class-tag: array,
  id: str
) -> array
```

items `dictionary`

The item tree.

class-tag `array`

The item’s class.

id `str`

The item’s ID.

get-item-name-id

Returns the name and label of the specified item.

Parameters

```
get-item-name-id(
  config: dictionary,
  items: dictionary,
  tag: array
) -> array
```

config `dictionary`

Full config.

items `dictionary`

Full item tree.

tag `array`

Full item tag, including its ID.

make-class

Generates a class.

Parameters

```
make-class(
  id: str,
  name: str,
  namer: function auto,
  identifier: function auto,
  fields: dictionary,
  classes: dictionary,
  origins: dictionary
) -> dictionary
```

id `str`

Class short identifier. Typically the first letter of the name, e.g. `"R"`.

name `str`

Class name.

namer `function` or `auto`

Function that gives a display name to the items of the class, of form `(class-tag: array, id: str, fields: dictionary, index: int, root-class-name: str, class-name: str) -> str`. If `auto`, it inherits from its ancestors (root classes can't set it as `auto`).

Default: `auto`

identifier `function` or `auto`

Function that gives an unique identifier to each item of the class, of form `(class-tag: array, id: str, fields: dictionary, index: int, root-class-name: str, class-name: str) -> str`. If `auto`, it inherits from its ancestors (root classes can't set it as `auto`).

Default: `auto`

fields `dictionary`

Set of fields that apply to the class. Generate them using `make-field`.

Default: `()`

classes `dictionary`

sub-classes belonging to this class. Fields belonging to this class are inherited by classes. Generate them using `make-class`.

Default: `()`

origins `dictionary`

List of classes that are the origin to this class. Note that **only** a “terminal class”, that is, a **class without classes** can have origins. Generate them using `make-origins`.

Default: `(:)`

make-config

Creates a configuration object from SRS classes.

Each class must be generated using `make-class`.

Parameters

```
make-config(
  item-formatter: function none,
  template-formatter: function none,
  traceability-formatter,
  language: str none,
  classes: array
) -> dictionary
```

item-formatter `function` or `none`

Default item formatter, of form `(class-tag: array, id: str, item: dictionary, index: int, config: dict, items: dictionary) -> content`.

Default: `none`

template-formatter `function` or `none`

Default template formatter, of form `(config: dict, tag: array, id: str) -> content`.

Default: `none`

language `str` or `none`

Default: `none`

classes `array`

Classes to use.

Default: `()`

make-enum-field

Encapsulates an enum field. These fields will receive a key as a value, and print its related value.

Example:

```
make-enum-field(
  h: "High",
  m: "Medium",
  l: "Low",
)
```

Parameters

```
make-enum-field(..values) -> dictionary
```

make-field

Generates a class field.

Parameters

```
make-field(
  name: str,
  value: dictionary str,
  description: content
) -> dictionary
```

name `str`

Field name.

value `dictionary` or `str`

Field values. Can be either an enumeration (`enum-field`) or content (`content-field`).

description `content`

Field description.

make-item

Creates an item belonging to the specified class. The specified fields are the ones belonging to the class.

Example:

```
make-item(
  "cool-req",
  ("R", "S", "NF"),
  origins: (("R", "U", "RE", "user-req")),
  Description: [ The software shall be cool. ],
  Necessity: "h",
  Priority: "h",
  Stability: "c",
  Verifiability: "l"
)
```

Parameters

```
make-item(
  id: str,
  class: array,
  origins: array,
  ..fields: arguments
) -> dictionary
```

id `str`

Item ID. This must be unique inside the class.

class `array`

Item class, expressed as the class hierarchy

origins `array`

Array of tags of items that give origin to this one.

Default: `()`

..fields `arguments`

Fields of the item, according to the class, e.g. `Name: "Potato"`.

make-origins

Generates an origins object.

Each origin is an array of `tags`. Generate them with `make-tag`.

Parameters

```
make-origins(
  description: content,
  ..tags
) -> dictionary
```

description `content`

Description, or justification of the origins.

make-tag

An unique identifier of an item and/or class, composed of its class path.

Each class is specified by the its ID.

Example:

```
make-tag("R", "U", "CA")
```

Parameters

```
make-tag(..path) -> array
```

show-items

Shows the items belonging to the specified class tag.

Parameters

```
show-items(
  reqs: dictionary,
  tag: array,
  formatter: function auto
) -> content
```

reqs `dictionary`

Requirements object.

tag `array`

Item class tag, use `make-tag` to generate it. Must be a terminal class.

formatter `function` or `auto`

Formatter function, of format `(class-tag: array, id: str, item: dictionary, index: int, config: dict, items: dictionary) -> content`. If `auto`, it uses the configuration's default `item-formatter`.

Default: `auto`

show-template

Shows a template for the items of the specified class tag.

Parameters

```
show-template(
  reqs: dictionary,
  tag: array,
  id: str,
  formatter: function auto
) -> content
```

reqs `dictionary`

Requirements object.

tag `array`

Item class tag, use `make-tag` to generate it.

id `str`

ID to give to the template, typically used in the label of a figure.

formatter `function` or `auto`

Formatter function, of format `(config: dict, tag: array, id: str) -> content`. If `auto`, it uses the configuration's default `template-formatter`.

Default: `auto`

tag-to-class-tree

This function returns an array of all the classes and subclasses of `config` following the path described by `tag`. For instance: if `tag` is `("R", "F")`, then the result will be an array of two elements: the class "R" and its child class "F".

Parameters

```
tag-to-class-tree(
  config: dictionary,
  tag: array
) -> array
```

config `dictionary`

The configuration lol

tag `array`

The tag

validate-config

Validates the configuration object.

It returns a result. That is, a pair `(ok, err)` where `ok` is the result of the operation and `err` is the error message in case of error.

Parameters

```
validate-config(config: dictionary) -> array
```

config `dictionary`

Configuration object. Generate it using `make-field`.

content-field `str`

Encapsulates a content field. These fields will receive a `content` object as a value.

srs.defaults

- field-namer-maker()
- identifier-maker()
- incremental-namer-maker()
- table-formatter()
- table-item-formatter-maker()
- table-template-formatter-maker()
- table-traceability-formatter-maker()
- traceability-table-formatter()

Variables

- base-classes
- simple-classes

field-namer-maker

Returns a namer function that names the item by the specified field name.

Parameters

```
field-namer-maker(field-name: str) -> function
```

field-name `str`

Class field to get the item name from.

identifier-maker

Returns a identifier function, which creates tags of form `<prefix><separator><id>`.

Parameters

```
identifier-maker(
  prefix: function str none,
  separator: str
) -> function
```

prefix `function` or `str` or `none`

Prefix to use. Can be dynamic, if a function `(tag: array, root-class-name: str, class-name: str, separator: str) -> str` is supplied, or static, if string, or `none`, in which case the tag will be just the ID.

Default: `none`

separator `str`

Separator between the prefix and name. If `prefix` is a function, this will be the argument passed.

Default: `"-"`

incremental-namer-maker

Returns a namer function that names the item with an incremental name, e.g. `<prefix><separator>0X`.

Parameters

```
incremental-namer-maker(
  prefix: function str none,
  separator: str,
  start: int,
  width: int,
  fillchar
) -> function
```

prefix `function` or `str` or `none`

Prefix to use. Can be dynamic, if a function `(tag: array, root-class-name: str, class-name: str, separator: str) -> str` is supplied, or static, if string, or `none`, in which case the tag will be just the ID.

Default: `none`

separator `str`

Separator between the prefix and name. If `prefix` is a function, this will be the argument passed.

Default: `"-"`

start `int`

Starting index.

Default: `1`

width `int`

Width of the index, which will be padded with zeroes.

Default: `2`

table-formatter

This function returns a labeled table with the specified `contents`.

The label will be `srs:<id>`.

Parameters

```
table-formatter(
  contents: array,
  id: str,
  caption: content str,
  language: str,
  breakable: bool,
  justify: array,
  style: dictionary
) -> content
```

contents `array`

The table's contents.

id `str`

Unique item ID, used in the label.

caption `content` or `str`

The table's caption

language `str`

Language to use.

breakable `bool`

Whether the table can span multiple pages.

Default: `false`

justify `array`

Justification of the two columns, e.g. (true, false)

Default: `(false, true)`

style `dictionary`

Parameters to pass to the table, e.g. `(columns: (1fr, 1fr), gutter: 1em)`

Default: `(columns: 2)`

table-item-formatter-maker

Returns an item formatter that formats the item as a table.

The table's label will have the form `srs:<tag>`, where `<tag>` is the result of calling `tagger`.

Parameters

```
table-item-formatter-maker(
  language: str auto,
  breakable: bool,
  justify: array,
  style: dictionary
) -> function
```

language `str` or `auto`

Language of the captions. If `auto`, it will use the one in `config.language`

Default: `auto`

breakable `bool`

If the table can be broken in several pages.

Default: `false`

justify `array`

Justification of the two columns, e.g. (true, false)

Default: `(false, true)`

style `dictionary`

Parameters to pass to the table, e.g. `(columns: (1fr, 1fr), align: left, gutter: 1em)`

Default: `(columns: 2)`

table-template-formatter-maker

Returns a template formatter that formats the template as a table.

The table's label will have the form `srs:<tag>`, where `<tag>` is the result of calling `tagger`.

Parameters

```
table-template-formatter-maker(
  language: str auto,
  breakable: bool,
  justify: array,
  style: dictionary
) -> function
```

language `str` or `auto`

Language of the captions. If `auto`, it will use the one in `config.language`

Default: `auto`

breakable `bool`

If the table can be broken in several pages.

Default: `false`

justify `array`

Justification of the two columns, e.g. (true, false)

Default: `(false, true)`

style `dictionary`

Parameters to pass to the table, e.g. `(columns: (1fr, 1fr), align: left, gutter: 1em)`

Default: `(columns: 2)`

table-traceability-formatter-maker

Returns a traceability matrix formatter that formats the relationship between two classes as a table.

This formatter will create a table that shows the relationships between the fields of the two classes, indicating which fields in the first class are related to which fields in the second class.

Parameters

```
table-traceability-formatter-maker(
  language: str auto,
  breakable: bool,
  marker: symbol,
  rotation-angle: angle,
  style: dictionary,
  column-size: length
) -> function
```

language `str` or `auto`

Language of the captions. If `auto`, it will use the one in `config.language`

Default: `auto`

breakable `bool`

If the table can be broken in several pages.

Default: `false`

marker `symbol`

Symbol to use for marking related fields.

Default: `sym.checkmark`

rotation-angle `angle`

Rotation angle for the table headers.

Default: `0deg`

style `dictionary`

Parameters to pass to the table, e.g. `(align: center, gutter: 0em)`

Default: `none`

column-size `length`

Size of the columns in the table.

Default: `auto`

traceability-table-formatter

This function returns a labeled table with the specified `contents` for an n*m sized matrix.

The label will be `srs:<id>`.

Parameters

```
traceability-table-formatter(
  contents: array,
  id: str,
  caption: content str,
  language: str,
  breakable: bool,
  rotation-angle: angle,
  displacement: length,
  style: dictionary,
  column-size: length
) -> content
```

contents `array`

The table’s contents.

id `str`

Unique item ID, used in the label.

caption `content` or `str`

The table’s caption

language `str`

Language to use.

breakable `bool`

Whether the table can span multiple pages.

Default: `false`

rotation-angle `angle`

Rotation angle for the table headers.

Default: `0deg`

displacement `length`

Displacement for the table headers (used normally in conjunction with rotate).

Default: `-0em`

style `dictionary`

Parameters to pass to the table, e.g. `(columns: (1fr, 1fr), gutter: 1em)`

Default: `none`

column-size `length`

Size of the columns in the table.

Default: `auto`

base-classes `array`

Base class set.

Includes:
- Requirement (R)
  - User Requirement (U)
    - Capabilities (CA)
    - Restrictions (RE)
  - Software Requirement (S)
    - Functional (FN)
    - Non-Functional (NF)
- Use Case (U)
- Component (C)
- Test (T)
  - Verification (VET)
  - Validation (VAT)

simple-classes `array`

Simple classes set.

Includes:
- Requirement (R)
  - User Requirement (U)
  - Software Requirement (S)
    - Functional (FN)
    - Non-Functional (NF)