SCHEMATA

Mariano Montone (marianomontone@gmail.com)

Copyright © 2021 Mariano Montone

Table of Contents

1 Introduction

SCHEMATA is a web forms handling library for Common Lisp.

Although it is potentially framework agnostic, it runs on top of Hunchentoot at the moment.

It features:

- Several form field types: String, boolean, integer, email, password fields. And more.
- Custom fields. SCHEMATA is extensible and it is possible to define new field types.
- Server and client side validation
- Rendering backends. Forms can be rendered via CL-WHO, or Djula, or something else; the backend is pluggable. The default renderer is CL-WHO.
- Themes (like Bootstrap)
- Control on rendering and layout.
- Handling of form errors.
- CSRF protection

2 Installation

With Quicklisp:

```
(ql:quickload "schemata")
```

When you want to use a form renderer such as `:who` or `:djula`, quickload the associated package: `schemata.who`, `schemata.who.bootstrap`, `schemata.djula`.

3 Usage

3.1 Basics

4 API

4.1 SCHEMATA package

SCHEMATA [PACKAGE]

External definitions

Macros

SCHEMATA:SCHEMA (*schema-def*) [Macro]
Wrapper macro for schema definitions.

SCHEMATA:DEFINE-SCHEMA (*name schema*) [Macro]
Register SCHEMA under NAME. The schema can then be accessed via FIND-SCHEMA.

Generic functions

SCHEMATA:PARSE-WITH-SCHEMA (*schema string-or-data*) [Generic-Function]
Parses the string to an association list using the schema

SCHEMATA:ATTRIBUTE-NAME (*sb-pcl::object*) [Generic-Function]

SCHEMATA:UNSERIALIZE-WITH-SCHEMA (*schema data format*) [Generic-Function]

SCHEMATA:SCHEMA-DOCUMENTATION (*sb-pcl::object*) [Generic-Function]

SCHEMATA:OBJECT-CLASS (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-PARSER (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-VALIDATOR (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-ADD-VALIDATOR (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-EXTERNAL-NAME (*sb-pcl::object*) [Generic-Function]

SCHEMATA:SCHEMA-TYPE (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-FORMATTER (*sb-pcl::object*) [Generic-Function]

SCHEMATA:OBJECT-NAME (*sb-pcl::object*) [Generic-Function]

SCHEMATA:OBJECT-ATTRIBUTES (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-ACCESSOR (*sb-pcl::object*) [Generic-Function]

SCHEMATA:ATTRIBUTE-TYPE (*sb-pcl::object*) [Generic-Function]

Functions

`SCHEMATA:ATTRIBUTE-READER` (*attribute*) [Function]

`SCHEMATA:POPULATE-WITH-SCHEMA` (*schema object data* **&key** *exclude*) [Function]

Populate CLOS objects from data + schema. Attributes members of EXCLUDE parameter are not populated.

`SCHEMATA:SCHEMA-CLASS-SCHEMA` (*schema-class*) [Function]

Generate a schema using the schema class meta info

`SCHEMATA:SERIALIZE-WITH-SCHEMA` (*schema input* **&optional** *(serializer generic-serializer::*serializer*) (stream generic-serializer::*serializer-output*)*) [Function]

`SCHEMATA:SCHEMA-SPEC` (*schema*) [Function]

`SCHEMATA:ATTRIBUTE-TYPE-NAME` (*attribute*) [Function]

`SCHEMATA:VALIDATION-ERROR` (*message* **&rest** *args*) [Function]

`SCHEMATA:VALIDATE-WITH-SCHEMA` (*schema data* **&key** *(collect-errors *collect-validation-errors*) (error-p *signal-validation-errors*)*) [Function]

Validate input using schema. Useful for validating resource operations posted content (for :post and :put methods). Input can be a string or an association list.

Args: - schema (symbol or schema): The schema - data (alist): The data to validate. - format (keyword): The data format. - collect-errors (boolean): If true, collect all the validation errors. If false, return the first validation error found. Default: true. - error-p (boolean): If true, when validation errors are found, a validation error is signaled. If false, the validation errors are returned as the function result and no error is signaled.

`SCHEMATA:ATTRIBUTE-OPTIONAL-P` (*attribute*) [Function]

`SCHEMATA:ATTRIBUTE-WRITER` (*attribute*) [Function]

`SCHEMATA:FIND-OBJECT-ATTRIBUTE` (*object attribute-name* **&key** *(error-p t)*) [Function]

`SCHEMATA:FIND-SCHEMA` (*name* **&optional** *(errorp t)*) [Function]

Find a schema definition by name

`SCHEMATA:PATCH-WITH-SCHEMA` (*schema object data*) [Function]

Populate CLOS objects from data + schema. Only populates attributes available in DATA, validating them. Useful for PATCH rest api operations implementations. DATA should be an association list.

Classes

`SCHEMATA:OBJECT-SCHEMA` [Class]

Class precedence list: `object-schema, schema, standard-object, t`

Slots:

- `name` — type: `(or string symbol)`; initarg: `:name`; reader: `schemata:object-name`; writer: `(setf schemata:object-name)`

  The name of the object.
- `attributes` — type: `list`; initarg: `:attributes`; reader: `schemata:object-attributes`; writer: `(setf schemata:object-attributes)`
- `class` — type: `(or null symbol)`; initarg: `:class`; reader: `schemata:object-class`; writer: `(setf schemata:object-class)`
- `ignore-unknown-attributes` — type: `boolean`; initarg: `:ignore-unknown-attributes`; reader: `schemata::ignore-unknown-attributes`; writer: `(setf schemata::ignore-unknown-att`
- `serializer` — type: `(or null trivial-types:function-designator)`; initarg: `:serializer`; reader: `schemata::object-serializer`; writer: `(setf schemata::object-serializer)`
- `unserializer` — type: `(or null trivial-types:function-designator)`; initarg: `:unserializer`; reader: `schemata::object-unserializer`; writer: `(setf schemata::object-unserializer)`

`SCHEMATA:SCHEMA-REFERENCE-SCHEMA` [Class]

Class precedence list: `schema-reference-schema, schema, standard-object, t`

Slots:

- `name` — type: `symbol`; initarg: `:schema-name`; reader: `schemata::schema-name`; writer: `(setf schemata::schema-name)`

`SCHEMATA:SCHEMA` [Class]

Class precedence list: `schema, standard-object, t`

Slots:

- `documentation` — type: `t`; initarg: `:documentation`; reader: `schemata:schema-documentation`; writer: `(setf schemata:schema-documentation)`

`SCHEMATA:VALIDATION-ERROR` [Class]

Class precedence list: `validation-error, error, serious-condition, condition, t`

`SCHEMATA:ATTRIBUTE` [Class]

Class precedence list: `attribute, schema, attribute-properties, standard-object, t`

Slots:

- `name` — type: `symbol`; initarg: `:name`; reader: `schemata:attribute-name`; writer: `(setf schemata:attribute-name)`
- `type` — type: `schemata:schema`; initarg: `:type`; reader: `schemata:attribute-type`; writer: `(setf schemata:attribute-type)`
- `accessor` — type: `(or null symbol)`; initarg: `:accessor`; reader: `schemata:attribute-accessor`; writer: `(setf schemata:attribute-accessor)`
- `writer` — type: `(or null trivial-types:function-designator)`; initarg: `:writer`

- `reader` — type: `(or null trivial-types:function-designator)`; initarg: `:reader`
- `slot` — type: `(or null symbol)`; initarg: `:slot`; reader: `schemata::attribute-slot`; writer: `(setf schemata::attribute-slot)`

`SCHEMATA:TYPE-SCHEMA` [Class]

Class precedence list: `type-schema, schema, standard-object, t`

Slots:

- `type` — type: `t`; initarg: `:type`; reader: `schemata:schema-type`; writer: `(setf schemata:schema-type)`

`SCHEMATA:SCHEMA-OBJECT` [Class]

Class precedence list: `schema-object, standard-object, t`

`SCHEMATA:SCHEMA-CLASS` [Class]

Metaclass for schema objects

Class precedence list: `schema-class, standard-class, class, specializer, metaobject, standard-object, t`

Slots:

- `schema-name` — type: `(or null string symbol)`; initarg: `:schema-name`; reader: `schemata::schema-name`; writer: `(setf schemata::schema-name)`

5 Index

(Index is nonexistent)

*

S