rest-server

A library for providing REST APIs
Release 0.1

by Mariano Montone

Table of Contents

This manual is for rest-server version 0.1.

1 Introduction

rest-server is a Common Lisp library for implementing REST APIs providers

1.1 Summary

rest-server is a Common Lisp library for implementing REST APIs providers

1.2 Installation

1.3 Feedback

Mail marianomontone at gmail dot com with feedback

1.4 Conventions

Hear are some coding conventions we'd like to follow:

- We *do* believe in documentation. Document your dynamic variables, functions, macros and classes. Besides, provide a documentation from a wider perspective. Provide diagrams and arquitecture documentation; examples and tutorials, too.
- Use widely known Common Lisp coding guidelines: `http://web.archive.org/web/20050305123711/www`

2 Overview

REST-SERVER is a Common Lisp library for implementing REST APIs servers.

Purpose of the library:

* Method matching - Based on HTTP method (GET, PUT, POST, DELETE) - Based on Accept request header - URL parsing (argument types) - Matching based on "extension": i.e. /users.json or /users.xml, etc - Method combinations?

* Serialization - Different serialization types (JSON, XML, S-expressions)

* Materialization (unserialization) - Types

* Error handling - Condition serialization - Error codes configuration

* Validation - Types - Schemas (JSON, XML schemas)

* Versioning Support for api versioning?

* Logging

* Cache handling

* Extensible - Backends (JSON, XML, etc) - Types - Validation

* Authentication - Different methods (token based, oauth) - Avoid changing the api interface spec because of this

* Modes - Debugging mode -> outputs full error serialization/backtrace - Production -> 500 internal server error

* Documentation - For the (lisp) developer - For the api consumer: https://github.com/mashery/iodocs http://swagger.wordnik.com/

* Resources - Good source of ideas: http://django-rest-framework.org/ http://www.restlet.org/

3 Example

```
(in-package :rest-server)

(defparameter *element*
  (element "user"
           (attribute "id" 22)
           (attribute "realname" "Mike")
           (attribute "groups"
                      (elements "groups"
                                (element "group"
                                         (attribute "id" 33)
                                         (attribute "title" "My group"))))))

(with-serializer-output t
  (with-serializer :json
    (serialize *element*)))

(with-output-to-string (s)
  (with-serializer-output s
    (with-serializer :json
      (serialize *element*))))

(cxml:with-xml-output (cxml:make-character-stream-sink t :indentation nil :omit-xml-de
  (with-serializer-output t
    (with-serializer :xml
      (serialize *element*))))

(with-output-to-string (s)
  (with-serializer-output s
    (with-serializer :xml
      (cxml:with-xml-output (cxml:make-character-stream-sink s :indentation nil :omit-
        (serialize *element*)))))

(with-serializer-output t
  (with-serializer :sexp
    (serialize *element*)))

(defpackage :api-test
  (:use :rest-server :cl))

(in-package :api-test)

(define-api api-test
  (:documentation "This is an api test"
   :content-types (list :json :xml))
```

```
  (get-users (:method :get
              :content-types (list :json)
              :uri-prefix "/users"
              :documentation "Retrive the users list")
             (&optional (expand-groups :boolean nil "Expand groups if true")))
  (get-user (:method :get
             :content-types (list :json)
             :uri-prefix "/users/id"
             :documentation "Retrive an user")
            ((id :string "The user id")
             &optional (expand-groups :boolean nil "Expand groups if true")))
  (create-user (:method :post
                :content-types (list :json)
                :uri-prefix "/users"
                :documentation "Create a user")
               ())
  (update-user (:method :put
                :content-types (list :json)
                :uri-prefix "/users/id"
                :documentation "Update a user")
               ((id :string "The user id")))
  (delete-user (:method :delete
                :content-types (list :json)
                :uri-prefix "/users/id"
                :documentation "Delete a user")
               ((id :string "The user id"))))

(defpackage :api-test-implementation
  (:use :cl :rest-server))

(in-package :api-test-implementation)

(defun get-users (&key (expand-groups nil))
  (list "user1" "user2" "user3" expand-groups))

(implement-api-function (get-user :serialization t)
    (id &key (expand-groups nil))
  (declare (ignore expand-groups))
  (element "user"
   (attribute "id" id)
           (attribute "groups"
                      (elements "groups"
                                (element "group"
                                         (attribute "id" 22)
                                         (attribute "name" "Group 1"))
                                (element "group"
                                         (attribute "id" 33)
```

```
                                        (attribute "name" "Group 2"))))))
```

```
(defun create-user (posted-content)
  (format nil "Create user: ~A" posted-content))

(defun update-user (posted-content id)
  (format nil "Update user: ~A ~A" id posted-content))

(defun delete-user (id)
  (format nil "Delete user: ~A" id))
```

4 System reference

`rest-server:attribute` [Class]
Class precedence list: `attribute, standard-object, t`
Serializer intermediate representation element attribute class

`rest-server:element` [Class]
Class precedence list: `element, standard-object, t`
Serializer intermediate representation element class

`rest-server:serializable-class` [Class]
Class precedence list: `serializable-class, standard-class, class, specializer, metaobject, standard-object, t`
Metaclass for serializable objects

`rest-server:add-list-member` *name value* **&key** *serializer stream* [Function]
Serializes a list member

`rest-server:attribute` *name value* [Function]
Build an element attribute to be serialized

`rest-server:element` *name* **&rest** *attributes* [Function]
Build an element to be serialized

`rest-server:elements` *name* **&rest** *elements* [Function]
Build a list of elements to be serialized

`rest-server:find-api` *name* [Function]
Find api by name

`rest-server:find-schema` *name* **&optional** *errorp* [Function]
Find a schema definition by name

`rest-server:make-api-function` *name method options args* [Function]
Make an api function.

`rest-server:serializable-class-schema` *serializable-class* [Function]
Generate a schema using the serializable class meta info

`rest-server:serialize-with-schema` *schema input* [Generic Function]
&optional *serializer stream*
Serialize input using schema

`rest-server:serialize` *element* **&optional** *serializer stream* [Generic Function]
Main serialization function. Takes the element to serialize, the serializer and the output stream

`rest-server:set-attribute` *name value* **&key** *serializer stream* [Function]
Serializes an element attribute and value

`rest-server:start-api-documentation` *api address port* [Function]
Start a web documentation application on the given api.

`rest-server:start-api` *api address port* **&optional** [Function]
api-implementation-package
Start an api at address and port. api-implementation-package: is the package where the api-functions are implemented.

`rest-server:define-api-function` *name method options args* [Macro]
Helper macro to define an api function

`rest-server:define-api` *name options* **&body** *functions* [Macro]
Define an api.

`rest-server:define-schema` *name schema* [Macro]
Define a schema

`rest-server:define-serializable-class` *name direct-superclasses* [Macro]
direct-slots **&rest** *options*
Helper macro to define serializable classes

`rest-server:implement-api-function` *name-and-options args* **&body** [Macro]
body
Define an api function implementation

`rest-server:with-api-backend` *backend* **&body** *body* [Macro]
Execute the client api function calling backend

`rest-server:with-api` *api* **&body** *body* [Macro]
Execute body under api scope. Example: (with-api test-api (define-api-function get-user :get (:url-prefix "users/{id}") '((:id :integer))))

`rest-server:with-attribute` *name* **&body** *body* [Macro]
Serializes an element attribute

`rest-server:with-element` *name* **&body** *body* [Macro]
Serializes a serializing element.

`rest-server:with-elements-list` *name* **&body** *body* [Macro]
Serializes an list of elements

`rest-server:with-list-member` *name* **&body** *body* [Macro]
Serializes a list member

`rest-server:with-serializer-output` *serializer-output* **&body** *body* [Macro]
Defines the serializer output when executing body.

Example:
(with-serializer-output s
(with-serializer :json (serialize user)))

`rest-server:with-serializer` *serializer* **&body** *body* [Macro]

Execute body in serializer scope. Binds *serializer* to serializer.

Example:
(with-serializer :json
(serialize user))

`rest-server:*development-mode*` [Variable]

Api development mode. One of :development, :testing, :production. Influences how errors are handled from the api

5 References

[Common Lisp Directory] [Common Lisp Wiki]

[Common Lisp Directory]: http://common-lisp.net [Common Lisp Wiki]: http://www.cliki.net

6 Index

6.1 Concept Index

6.2 Class Index

6.3 Function / Macro Index

6.4 Variable Index