

Configurator

A configuration management utility
User manual
Release 0.1

by Mariano Montone

Table of Contents

1	Introduction	1
1.1	Summary	1
1.2	Installation	1
1.3	Feedback	1
1.4	Conventions	1
2	Overview	2
3	Running	3
3.1	Running overview	3
3.2	Running modes	3
3.3	Command line summary	6
4	Configuration schemas	8
4.1	Schema sections	9
4.2	Schema options	10
4.2.1	Options dependencies	11
4.3	Built-in option types	12
4.3.1	String	12
4.3.2	Number	12
4.3.3	Boolean	12
4.3.4	Email	13
4.3.5	Url	13
4.3.6	Filename	13
4.3.7	Choice	13
4.3.8	List	13
5	Configurations	14
6	Examples	15
6.1	Use cases	17
6.1.1	Debugging	17
6.1.2	Logging	17
6.1.3	Testing	17
6.1.4	Deployment	17
7	Frontend	18
8	Custom option types	19

9	Language bindings.....	20
10	System reference	21
11	References	22
12	Index.....	23
12.1	Concept Index.....	23
12.2	Class Index.....	23
12.3	Function Index.....	23
12.4	Variable Index.....	23

This manual is for Configurator version 0.1.

Copyright © 2013 Mariano Montone

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being “A GNU Manual,” and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License.”

(a) The FSF’s Back-Cover Text is: “You have the freedom to copy and modify this GNU manual. Buying copies from the FSF supports it in developing GNU and promoting software freedom.”

This document is part of a collection distributed under the GNU Free Documentation License. If you want to distribute this document separately from the collection, you can do so by adding a copy of the license to the document, as described in section 6 of the license.

1 Introduction

Configurator is a configuration management utility.

You can get a copy and this manual at <http://mmontone.github.io/configurator>

1.1 Summary

Configurator is a configuration management utility

1.2 Installation

Download Configurator, or clone the git repository from <https://github.com/mmontone/configurator>. Then run `sudo sh ./install.sh`. That installs configurator on `/usr/local/lib/configurator` directory, and the configurator binary on `/usr/local/bin`. After that, you can run configurator typing `configurator` at the shell.

1.3 Feedback

Mail marianomontone@gmail.com with feedback

1.4 Conventions

Here are some coding conventions we'd like to follow

- We *do* believe in documentation. Document your variables, functions, macros and classes. Besides, provide a documentation from a wider perspective. Provide diagrams and architecture documentation; examples and tutorials, too. Consider using an automatic documentation generator (see the `bitacora` package in the dependencies).
- We don't want functions to be shorter than they should nor longer than they should. There is no "every function should have at most ten lines of code" rule. We think that coding is like literature to a great extent. So you should strive for beauty and clarity.

2 Overview

Configurator is a configuration management utility. It is written in Python and provides a Tk GUI at the moment.

The idea is to define configuration schemas and get a proper way of:

- Sharing and versioning your project's configuration schemas, but not your configurations. That way, you avoid overwriting configurations from different developers. Each developer has his own configurations that need to match the configuration schemas in the project. Whenever a project's configuration schema changes, each developer is responsible of updating his configurations to match the new schemas.
- Being able to define configuration schemas from the GUI, with no need for programming for most cases.
- Provide configurations documentation and validation.
- Edit configurations from a GUI.
- Define your own option configurations types and provide validation for them.

3 Running

3.1 Running overview

Configurator is run invoking *configurator* command from the command line. By default, it runs in normal mode; that means, it opens a GUI for adding, removing and editing configurations.

A configurations schemas files is required. By default, Configurator looks for *configurator.schema* in the current directory. It shows and errors if it can not find it. A different file or location can be specified through the *-schemas SCHEMAS* option.

If the schemas file is found, then it is parsed and loaded. The schemas file is in XML format.

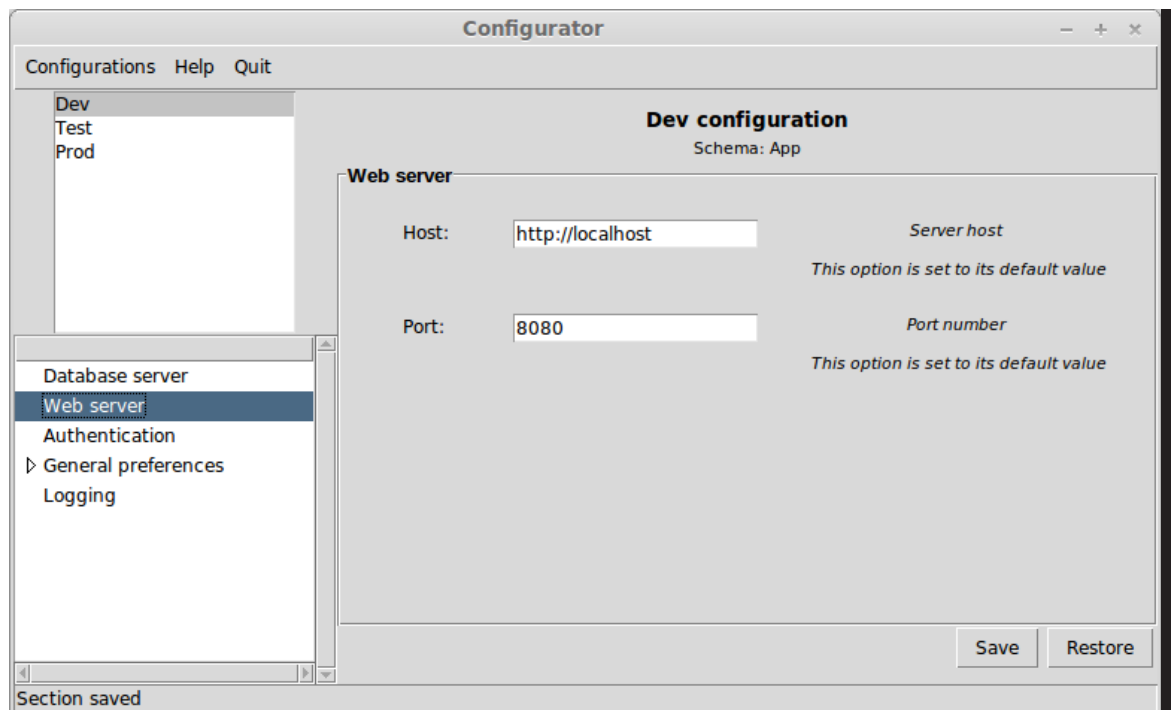
Apart from that, Configurator maintains configurations in another file, which by default is *configurator.config*. It can be specified to be something else through the *-configs CONFIGS* option.

3.2 Running modes

Configurator can be run in three different modes fundamentally.

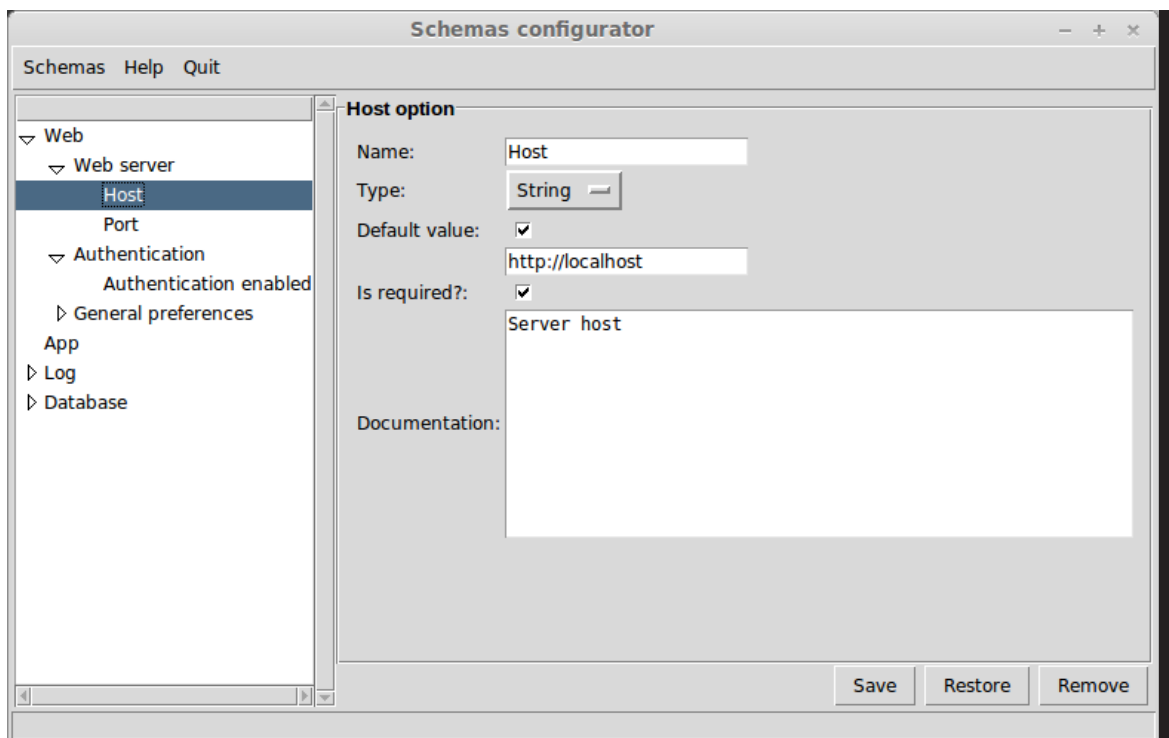
- *Normal mode*: this mode is invoked running configurator with no special arguments from the command line (apart from the schema and configs arguments). In this mode, the standard configuration navigation UI is opened. This UI is meant for end users. The user can create, remove and edit his configurations from here. He doesn't need to know how to build a configuration schema (although that is not difficult at all, as we will see.) Apart from that, when editing the configuration, the user gets a (hopefully) decent UI with custom option editors depending on the type of options and validation.

This is an example of Configurator running in normal mode:



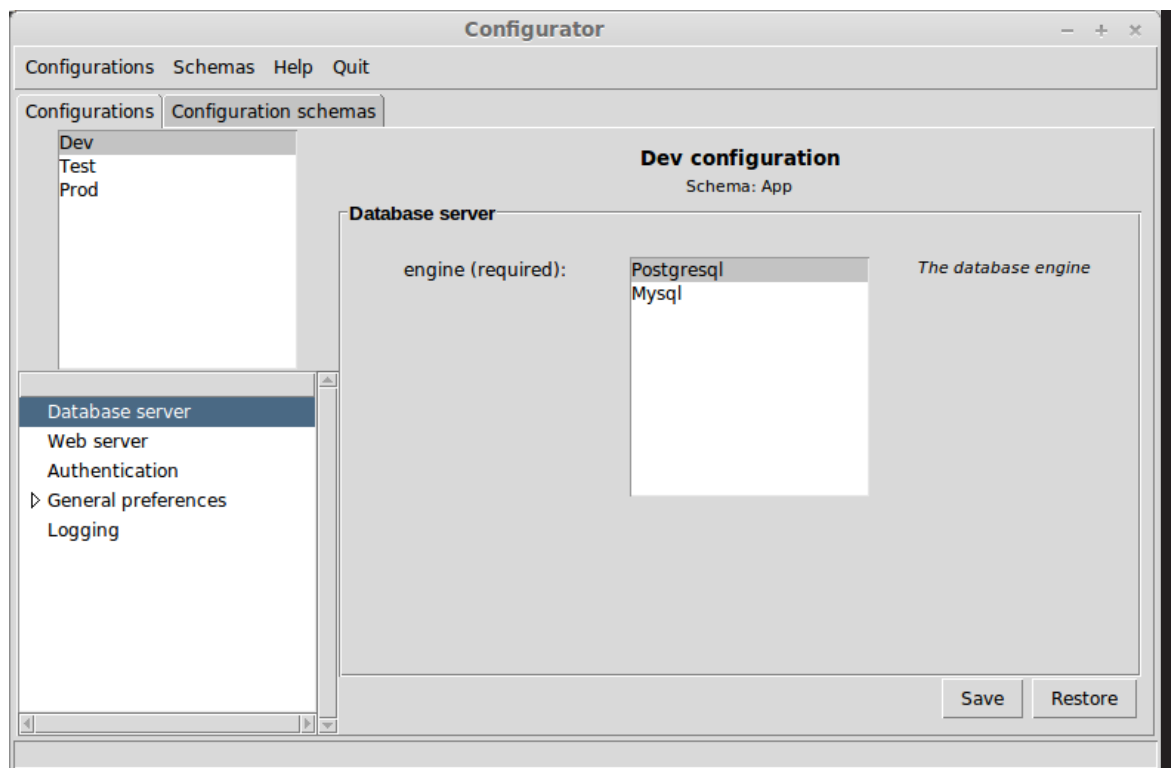
- *Setup mode*: this mode is invoked running configurator with the `-setup` option from the command line. In this mode, the configuration schemas navigator UI is opened. The developer can create, remove and edit configuration schemas from here. Configuration schemas are descriptions of how configurations should be, with nested sections and different type of options. He can build the application specific configuration schemas from here.

This is an example of Configurator running in setup mode:



- *Full mode*: this mode is invoked running configurator with the *-full* option from the command line. In this mode, both the configurations navigator and the configurations schemas navigator are available in two different tabs.

This is an example of Configurator running in full mode:



3.3 Command line summary

```
$> configurator -h
```

```
usage: configurator [-h] [-f] [-s SCHEMAS] [-c CONFIGS] [-l]
                  [-i INSPECT_CONFIG] [-g GET] [--set SET] [--config CONFIG]
                  [--validation VALIDATION] [--validate VALIDATE]
                  [--validate-all] [--json] [--setup] [--debug DEBUG]
```

Configurator. Configuration management utility.

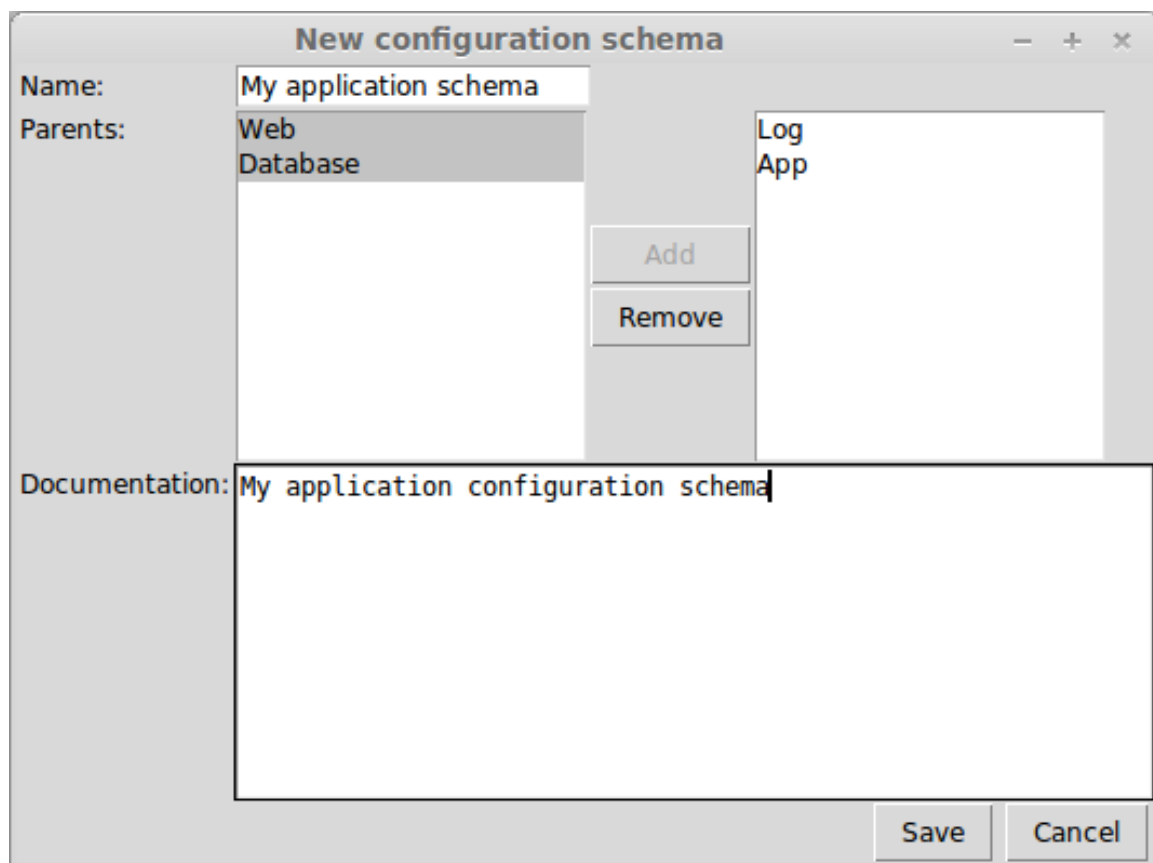
optional arguments:

```
-h, --help            show this help message and exit
-f, --full            Run the full configurator (both configurations and
                    schemas navigation)
-s SCHEMAS, --schemas SCHEMAS
                    The configuration schemas files. Default is
                    configurator.schema
-c CONFIGS, --configs CONFIGS
                    The configurations file. Default is
                    configurator.config
```

```
-l, --list-configs    List configurations
-i INSPECT_CONFIG, --inspect-config INSPECT_CONFIG
                    Inspect a configuration. A CSV(Comma separated values)
                    list with <option path>, <value>, <option type>,
                    <origin>
-g GET, --get GET    Get an option value
--set SET            Set an option value
--config CONFIG      Edit a specific configuration
--validation VALIDATION
                    Enable or disable configurations validation
--validate VALIDATE  Validate a configuration. Pass the configuration name
--validate-all      Validate all configurations
--json              Use JSON for communication
--setup             Edit configuration schemas
--debug DEBUG        Run in debug mode. Provide the debugging level, one of
                    DEBUG or INFO
```

4 Configuration schemas

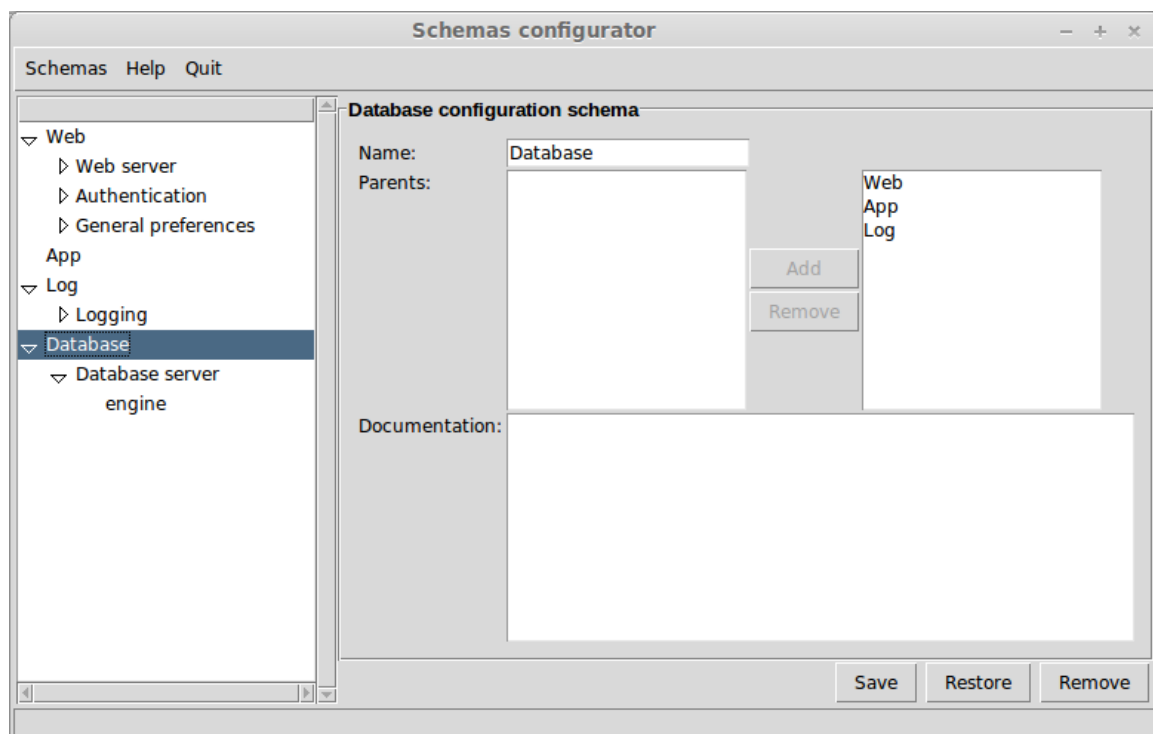
Configuration schemas define the configurations structure. They have a name, a list of parents, and a list of sections with options definitions.



The screenshot shows a dialog box titled "New configuration schema". It has a "Name:" field with the text "My application schema". Below it, the "Parents:" section contains a list with "Web" and "Database". To the right of this list is a larger text area containing "Log" and "App". Between the parents list and the right text area are "Add" and "Remove" buttons. At the bottom of the dialog is a "Documentation:" field with the text "My application configuration schema". The "Save" and "Cancel" buttons are at the bottom right.

Configuration schemas can be composed by inheriting from multiple parents. Configuration sections from the parents appear in the child configuration schema. For instance, a full stack web framework configuration schema could inherit from a generic Web schema for web server configuration, and another Database schema for database connection configuration.

Configuration schemas have sections, each containing other sections and schema options. The schemas sections and options can be manipulated in the tree appearing on the left of the configuration schemas navigator.

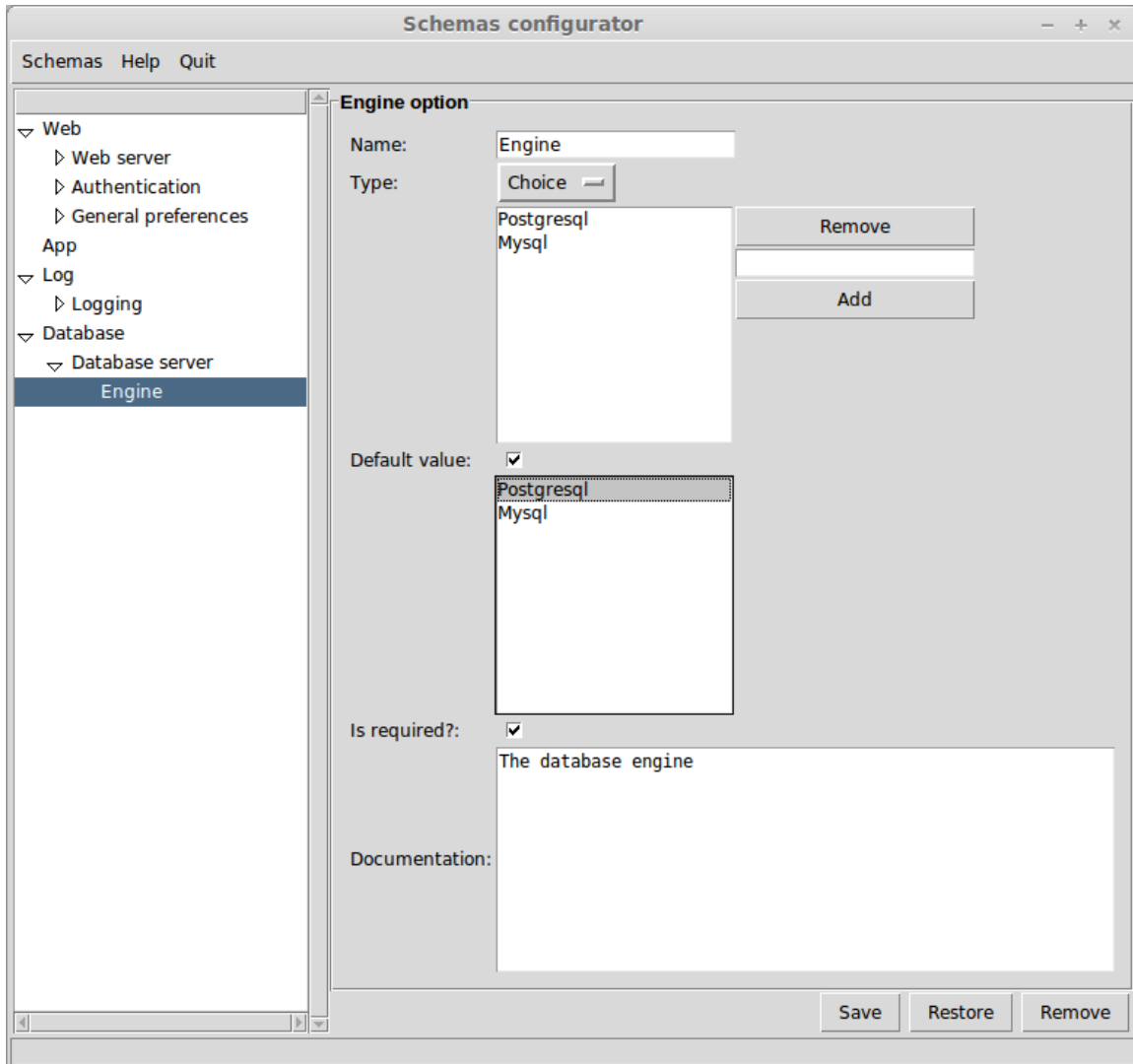


4.1 Schema sections

Each configuration schema section has a name, a documentation, subsections and a list of option schemas. Sections, subsections, and sections' options can all be added and removed from the tree widget on the left of the schemas navigator.

4.2 Schema options

Options in schemas have a name and a type.



The type of option determines the possible values that can be assigned to it in configurations. The option type can be String, Number, Boolean, Email, URI, Filename, Directory, Color, Timezone, Language, Country, Currency, Date, Time, Datetime, etc. Some of them will be described later. When editing configurations, each option is edited with its corre-

sponding editor, depending on the type of option. For instance, options of type Date are edited using a calendar.

Expire (required):

<< < December 2013 > >>

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Expiration

0 0 0

Apart from name and type, schema options specify if it required for the option to be assigned a value in the configuration. Also, they can have a default value in case the user doesn't assign one in the configuration.

They also have a documentation string. This is very important for the end user to know the option meaning.

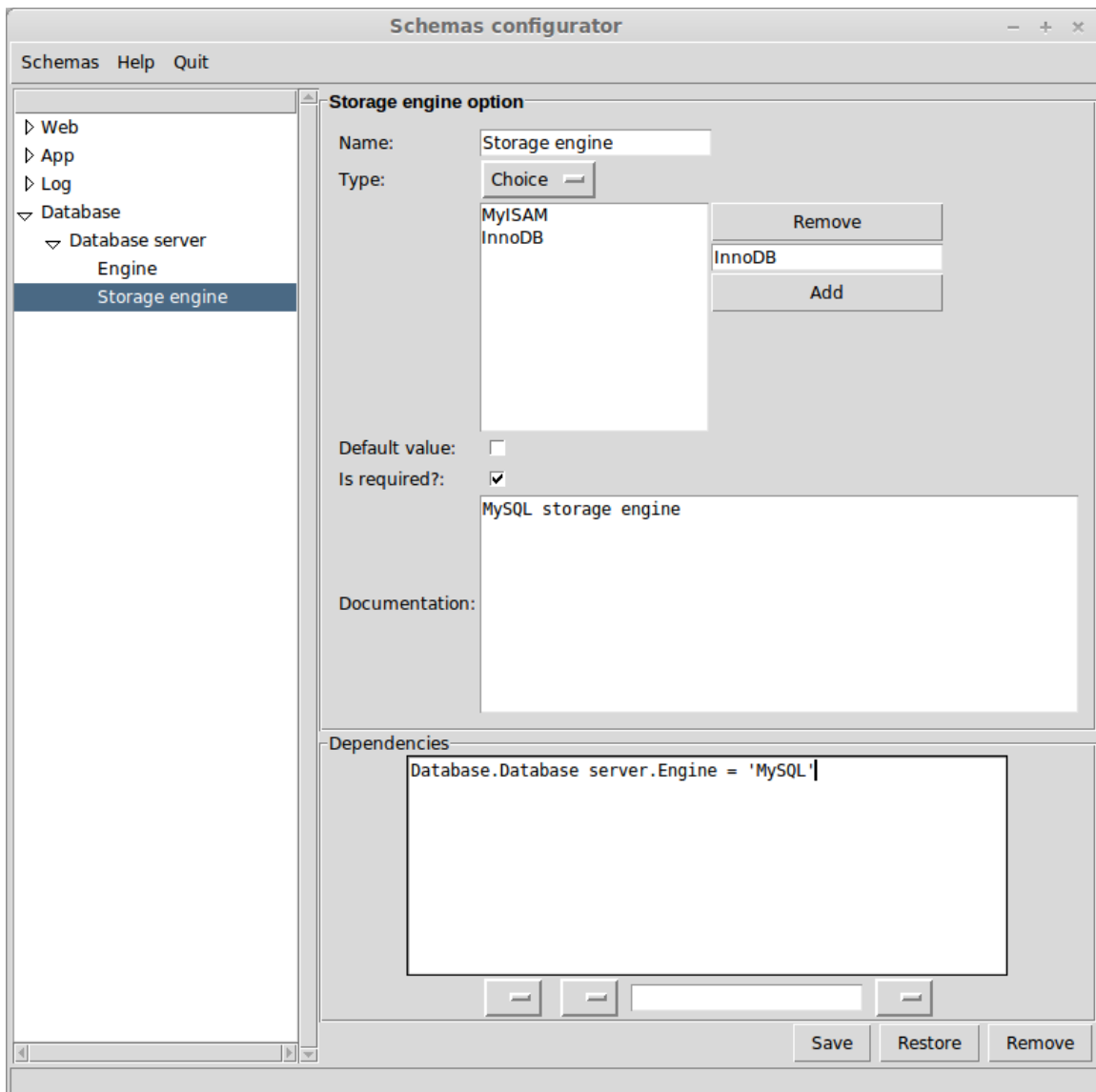
4.2.1 Options dependencies

It often happens that some options only make sense when some others are set to certain values. For instance, if we are configuring a database, then specific database engine options should only be available when the the specific engine is selected. Or a list of options make sense only when a boolean option is enabled.

Configurator has support for that by attaching a dependencies expression to an option schema. The dependency expression is written in a very simple domain specific language. They are basically boolean expressions, with options being referenced.

Example: MySQL supports two storage engines, MyISAM and InnoDB, but this option only makes sense for MySQL. To specify that, we can use the following dependencies expression: `Database.Engine = 'MySQL'`. This way the `Storage engine` option will only be editable when the `Database.Engine` is set to `'MySQL'`.

Dependency expressions can be added in the schema option editing screen:



4.3 Built-in option types

4.3.1 String

The String option type ensures that the the option value is of type string.

4.3.2 Number

The Number option type ensures that the the option value is of type Number.

4.3.3 Boolean

The boolean option type ensures that the the option value is of type boolean (True or False).

4.3.4 Email

The email option type ensures that the the option value is a valid email string.

4.3.5 Url

The url option ensures that the the option value is a valid url.

4.3.6 Filename

The pathaname option type ensures that the the option value is a valid pathname and the file or directory exists.

4.3.7 Choice

The Choice option type ensures that the option value is one of the options listed.

4.3.8 List

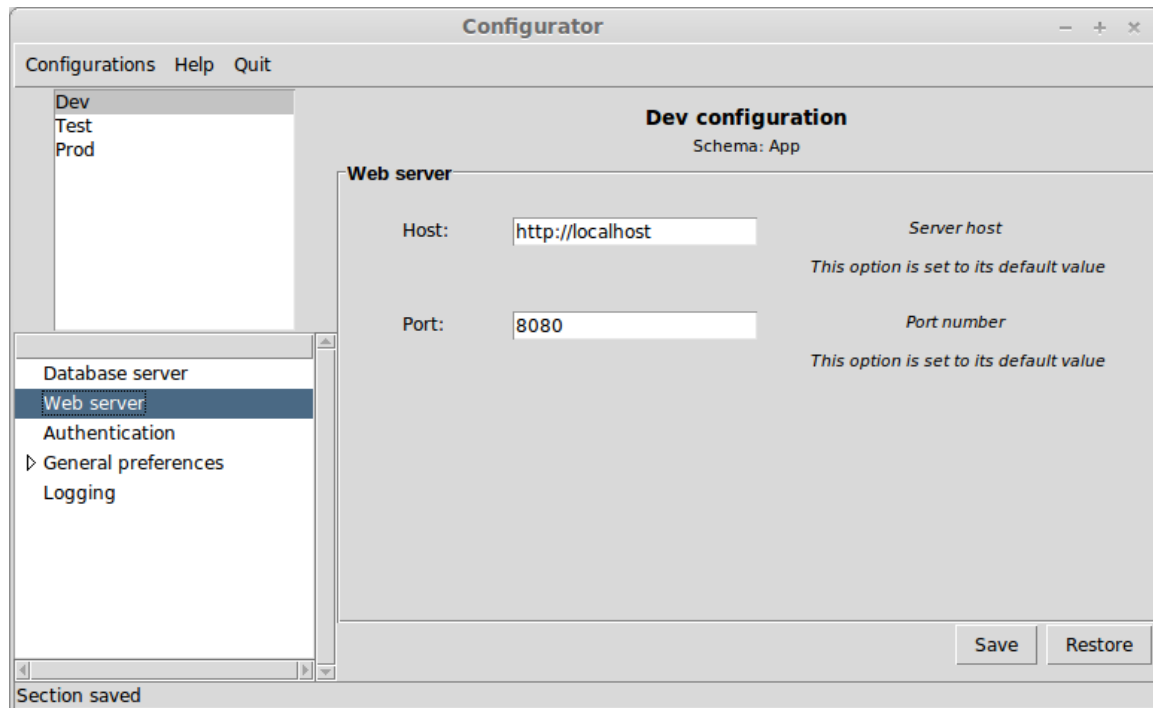
The *List* option type ensures that the option value is a subset of the options listed.

5 Configurations

Configurations are instances of Configuration schemas, much like objects are instances of classes in object oriented programming languages. That is, configurations structure is determined by the configuration schema they belong to. In particular, their sections are that of their configuration schemas; and their options values depend on the option schemas defined in the configuration schemas.

Configurations belong to one and only one configuration schema, and that's compulsory, as it can be expected. Besides, configurations can inherit from each other: a configuration can inherit from another configuration and overwrite its parent options values. A configuration can have one and only one parent, or no parent; this is different from configuration schemas, that can have several parents. An understandable restriction is that the configuration parent schema has to be the same that the configuration schema.

Configurations can be added and removed from the list appearing on the left of the configurations navigator.



Configurations can be loaded and saved. They are serialized in XML format. The default filename is `configurator.config`, but it can be changed if desired.

Configuration options editing happens on the right panel of the configurations navigator. A specific option editor is offered for each type of option, and each option documentation is displayed too. When trying to save a configuration section, it is ensured that required options (options declared with `required` enabled in the configuration schema) are filled. Options that are not currently set have their default value, if any. Also, options can be `set` and `unset`. Setting a configuration option means setting the configuration option in the current configuration to the value being shown in the option editor. Unsetting a configuration option means removing the option value setting from the current configuration.

6 Examples

Schemas definitions:

```
<schemas>
  <schema name="Web">
    <documentation></documentation>
    <section name="Web server">
      <documentation></documentation>
      <option name="Host">
        <documentation>Server host</documentation>
        <type name="String"/>
        <required>True</required>
        <default>http://localhost</default>
      </option>
      <option name="Port">
        <documentation>Port number</documentation>
        <type name="Number"/>
        <required>True</required>
        <default>8080</default>
      </option>
    </section>
    <section name="Authentication">
      <documentation></documentation>
      <option name="Authentication enabled">
        <documentation>Enable authentication?</documentation>
        <type name="Boolean"/>
        <required>False</required>
      </option>
    </section>
    <section name="General preferences">
      <documentation></documentation>
      <option name="Font size">
        <documentation>Font size</documentation>
        <type name="Number"/>
        <required>True</required>
      </option>
      <section name="Colors">
        <documentation></documentation>
        <option name="Background color">
          <documentation>Background color</documentation>
          <type name="Color"/>
          <required>True</required>
        </option>
      </section>
    </section>
  </schema>
  <schema name="App">
```

```

    <documentation></documentation>
    <parent name="Database"/>
    <parent name="Web"/>
    <parent name="Log"/>
</schema>
<schema name="Log">
    <documentation></documentation>
    <section name="Logging">
        <documentation></documentation>
        <option name="Logfile">
            <documentation>Where the logging happens</documentation>
            <type name="Filename"/>
            <required>True</required>
        </option>
        <option name="Expire">
            <documentation>Expiration</documentation>
            <type name="Datetime"/>
            <required>True</required>
        </option>
    </section>
</schema>
<schema name="Database">
    <documentation></documentation>
    <section name="Database server">
        <documentation></documentation>
        <option name="engine">
            <documentation>The database engine</documentation>
            <type name="Choice">
                <option value="Postgresql"/>
                <option value="Mysql"/>
            </type>
            <required>True</required>
        </option>
    </section>
</schema>
</schemas>

```

Configurations definitions:

```

<configurations>
    <configuration name="Dev">
        <schema name="App"/>
        <option path="Database.Database server.engine" value="Postgresql"/>
    </configuration>
    <configuration name="Test">
        <schema name="App"/>
        <parent name="Dev"/>
        <option path="Database.Database server.engine" value="Mysql"/>
    </configuration>
</configurations>

```

```
</configuration>
<configuration name="Prod">
  <schema name="Web"/>
  <option path="Web.Authentication.Authentication enabled" value="True"/>
  <option path="Web.General preferences.Colors.Background color" value="#6ed9d9"/>
</configuration>
</configurations>
```

6.1 Use cases

6.1.1 Debugging

6.1.2 Logging

6.1.3 Testing

6.1.4 Deployment

7 Frontend

Configurations can be edited from a Tk interface.

8 Custom option types

How to define custom option types

9 Language bindings

There are language bindings implemented for Python, PHP and Common Lisp for the moment.

Bindings can be found in the bindings directory. For how they are used, look at the tests in the corresponding directory.

In general, bindings invoke configurator from the command line using JSON for communication.

10 System reference

11 References

12 Index

12.1 Concept Index

B

boolean 12

C

choice 13
configuration schema 8
conventions 1

D

debugging 17
dependency 11
deployment 17

E

email 13
examples 15

F

feedback 1
filename 13

I

installation 1
introduction 1

L

list 13
logging 17

N

number 12

O

option 10
option type 12
overview 2

R

reference 22
running 3

S

section 9
string 12
summary 1

T

testing 17

U

url 13

12.2 Class Index

(Index is nonexistent)

12.3 Function Index

(Index is nonexistent)

12.4 Variable Index

(Index is nonexistent)