Create software configuration tools with Config : :Model

Dominique Dumont

Hewlett-Packard

SCI & OSL Technical Excellence Symposium 09

Outline

Configuration is often painful !

Configuration is often difficult for a user :

- Edit a text file outside of /home
- Read man pages
- Ensure consistency
- Even more difficult during package upgrades

Objective 1 : Make configuration easy for users

Provide a graphical interface with :

- Integrated help
- Default values
- Validation of configuration data
- Several levels of skills *(from beginner to master)*
- Search

Handle configuration upgrade smoothly (mostly no interaction)

Objective 1 : Make configuration easy for users

Provide a graphical interface with :

- Integrated help
- Default values
- Validation of configuration data
- Several levels of skills *(from beginner to master)*
- Search

Handle configuration upgrade smoothly (mostly no interaction)

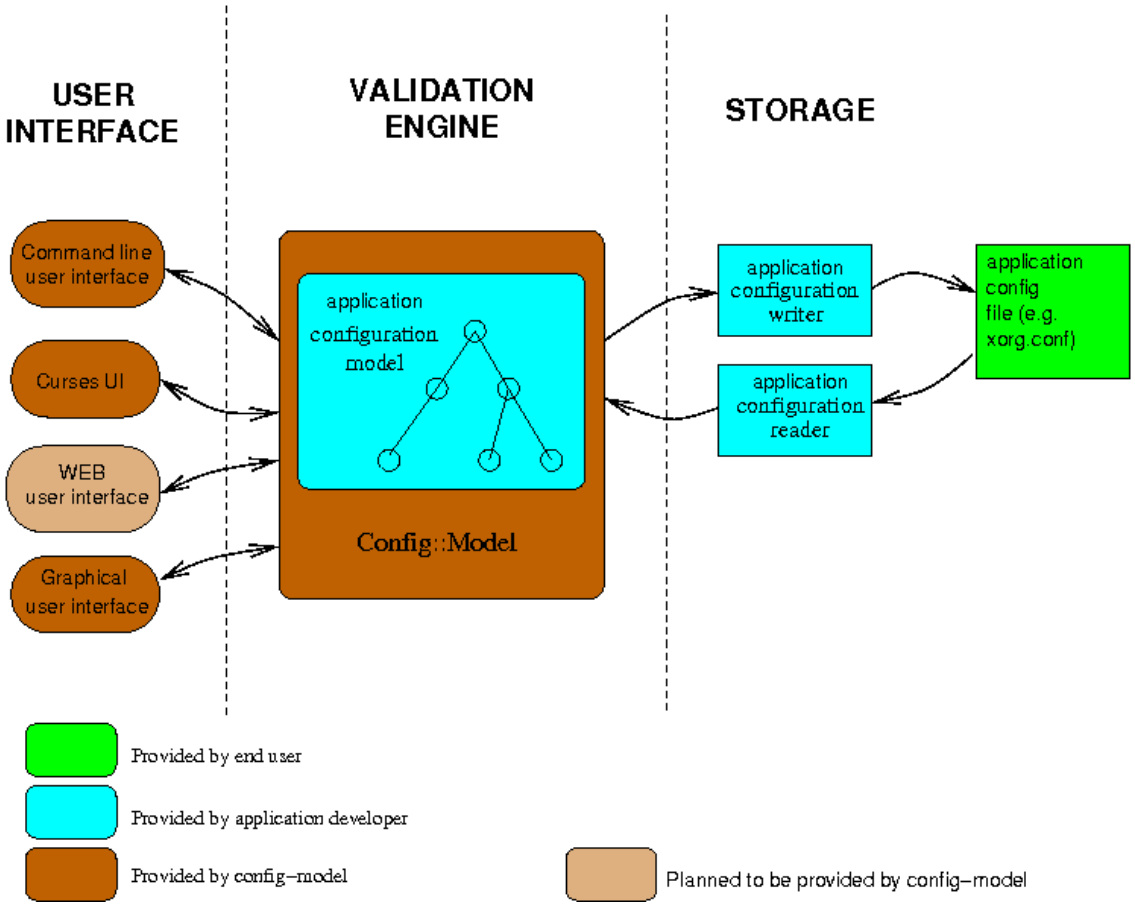
Objectif 2 : Make maintenance easy for developers

- The configuration validation must be easy to maintain :
  - Avoid ad-hoc validation code (e.g. don't rewrite Webmin)
  - Base validation on "meta-data" : the configuration model
  - Generate interfaces (graphicals or not) from the model
  - Cater for configuration upgrade
- Minimise code required to read or write configuration files :
  - Use existing libraries (Config : :Ini, Config : :Augeas – and all Augeas lenses...)
  - Provide basic classes to help configuration reads and writes

Objectif 2 : Make maintenance easy for developers

- The configuration validation must be easy to maintain :
  - Avoid ad-hoc validation code (e.g. don't rewrite Webmin)
  - Base validation on "meta-data" : the configuration model
  - Generate interfaces (graphicals or not) from the model
  - Cater for configuration upgrade
- Minimise code required to read or write configuration files :
  - Use existing libraries (Config : :Ini, Config : :Augeas – and all Augeas lenses...)
  - Provide basic classes to help configuration reads and writes

Config : :Model design



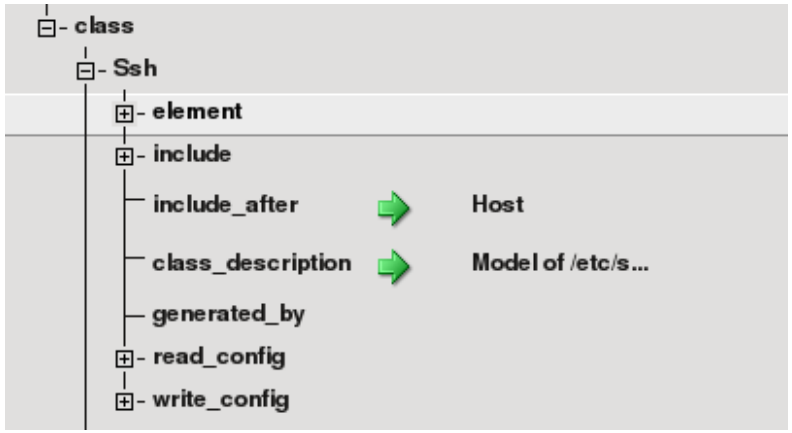
What is a model ?

A model defines a tree structure :

- A class is represented by a node
- A parameter is represented by a leaf

Each class contains :

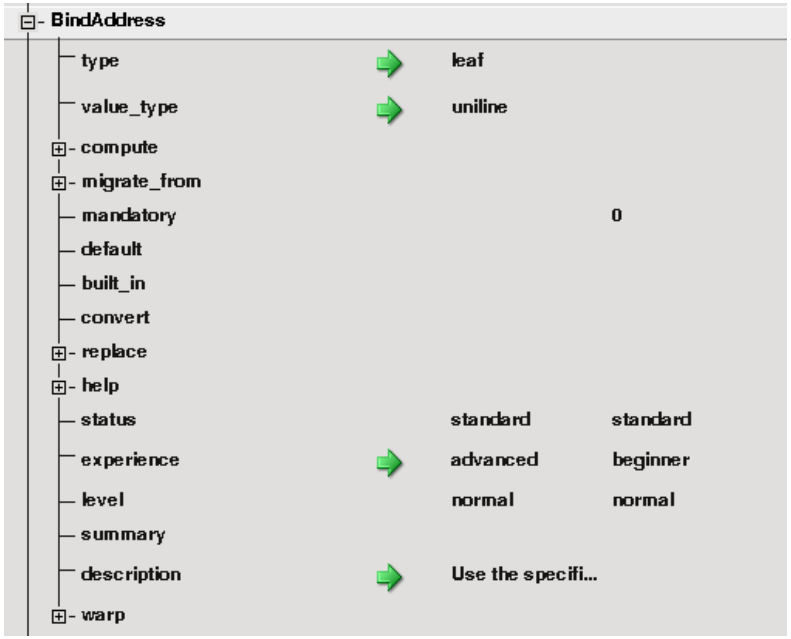
- a set of elements (parameters)
- optional : a specification to access configuration file (backend)



Simple elements

Each element has :

- a type (leaf, hash, list, node)
- constraints (integer, max, mini ...)
- a default value
- a description and a summary (for integrated help)
- an experience level (beginner, advanced, master)
- a status (normal or obsolete)



Complex elements

A configuration model can also define interactions between elements :

- Model warp (example : Xorg driver options change depending on declared driver) (*warp*)
- Simple computation from other elements (used for upgrades) (*compute* and *migrate_from*)
- References (example : Xorg : :Device : :Radeon, Monitor-DVI-0 must refer to one of the monitors declared in Monitor section)

Model analysis

- Read the application man pages :
  - Find the structure
  - Identify configuration parameters, their constraints and relations
- Find several valid examples :
  - To verify that the documentation was understood
  - For the non-regression test

Model declaration

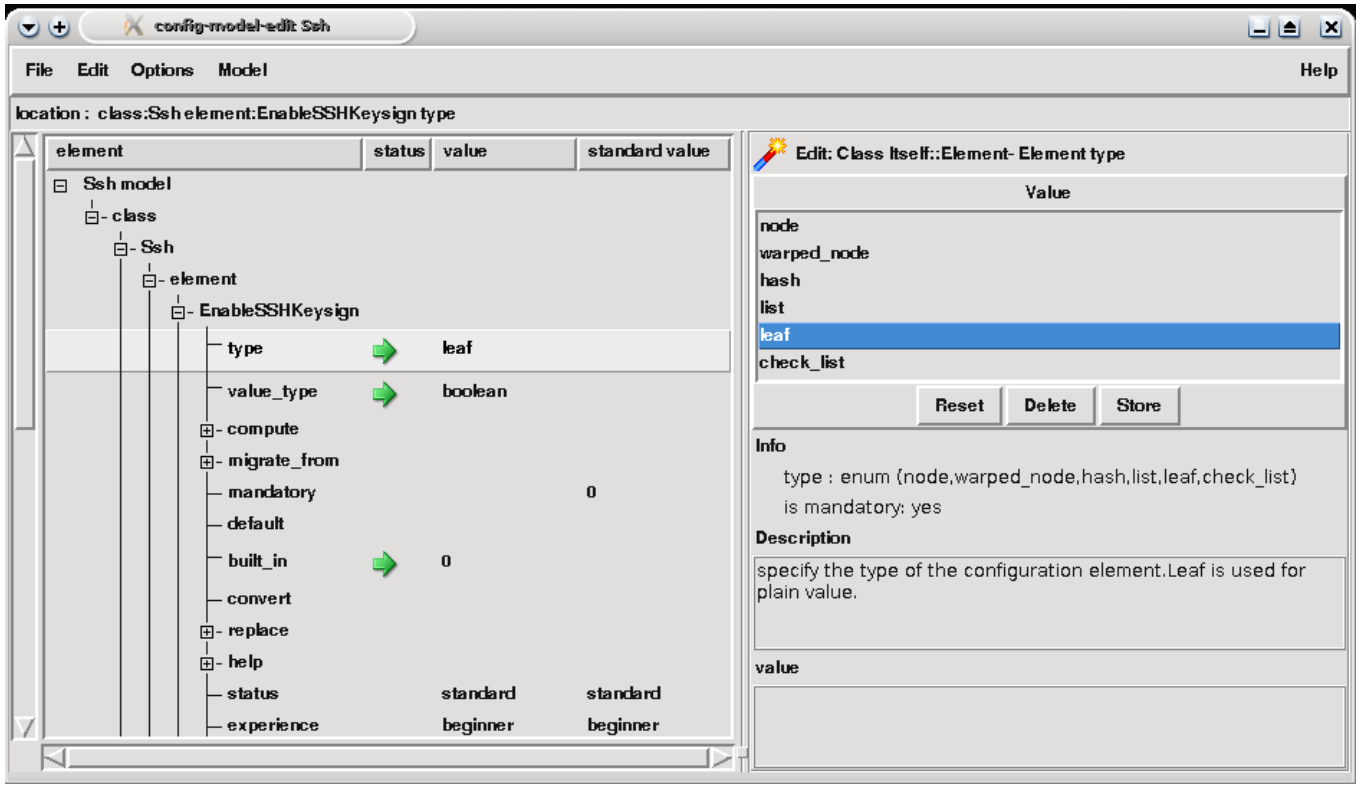
In summary, configuration documentation is translated in a format usable by Config : :Model :

- The structure is translated into configuration classes
- Configuration parameters into elements
- Constraints into element attributes

```
name => 'Ssh',                  # class name
element => [
  EnableSSHKeysign => {         # element name
    type => 'leaf',
    value_type => 'boolean',
    built_in => '0',            # default value
    description => 'Setting ...',
  },
]
```

Declaration (easier mode)

Since writing a data structure is not fun (even with Perl), a model can be create with a GUI :



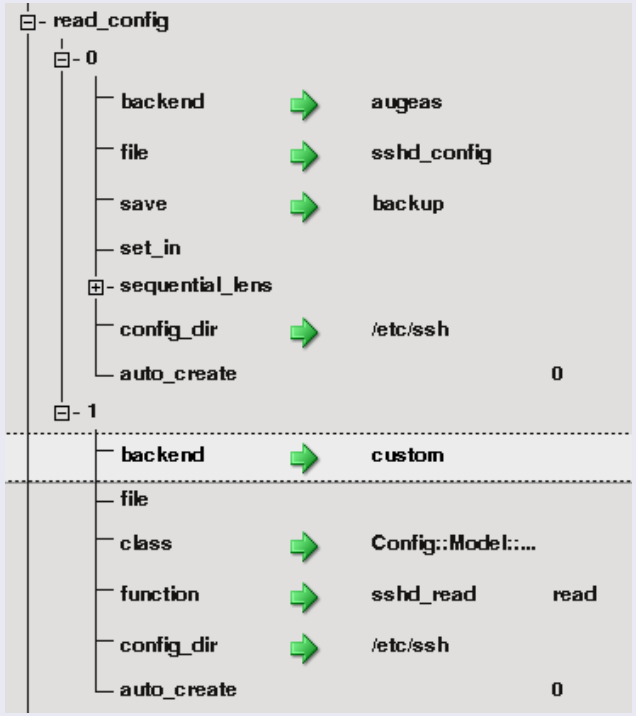
From time to time, do a Menu → Model → test

Reading configuration files

In the model

- Declare the mechanism (*backend*)
  - Built-in (Perl file, Ini file...)
  - Plug-in (Backend class)
  - custom → call-back must also be provided
- Mechanism parameters
- Specifications are tried in order

Example



Writing configuration files

In the model

- Not needed if write specification is the same as read
- Same parameters as read spec
- Tried in order until first success

Note

With these specifications, configuration can be migrated from one syntax to another.

Example

```
write_config => [
 {
  backend     => 'augeas',
  save        => 'backup',
  config_dir  => '/etc/ssh',
  file        => 'sshd_config',
 },
 {
  backend    => 'custom',
  class      => 'C::M::OpenSsh',
  function   => 'sshd_write',
  config_dir => '/etc/ssh'
 }
],
```

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade

Prepare configuration updates

For smooth upgrades

For application designers :

1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

For more information on migration applied to software packages, see http://wiki.debian.org/PackageConfigUpgrade

Prepare configuration updates

For smooth upgrades

For application designers :

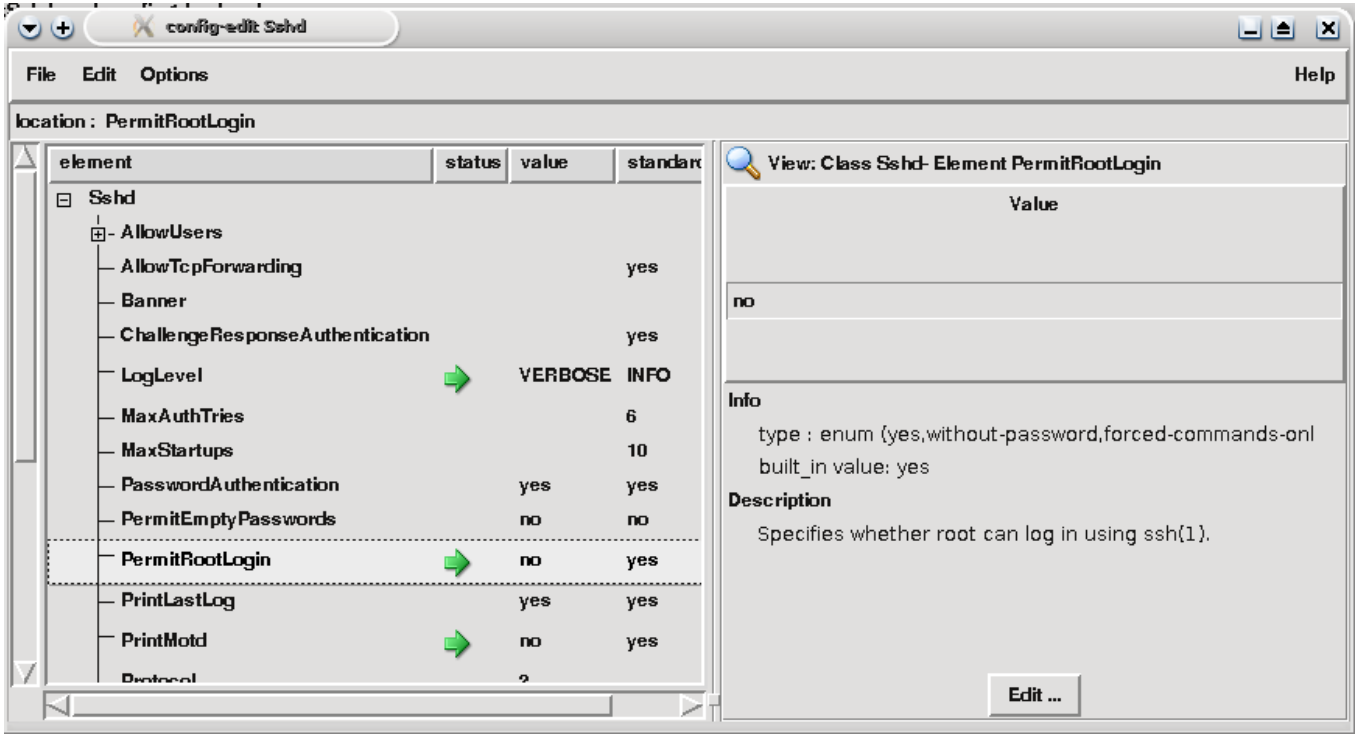
1. No new parameters <-> no new problems
2. Picking parameter name and value : A good name is better than 3 pages of doc
3. Default values : Application can work even with an empty configuration file

But, if needed, a model can specify :

- How to replace a value replace
- Obsolete parameters status
- How to migrate a value migrate with a computation formula

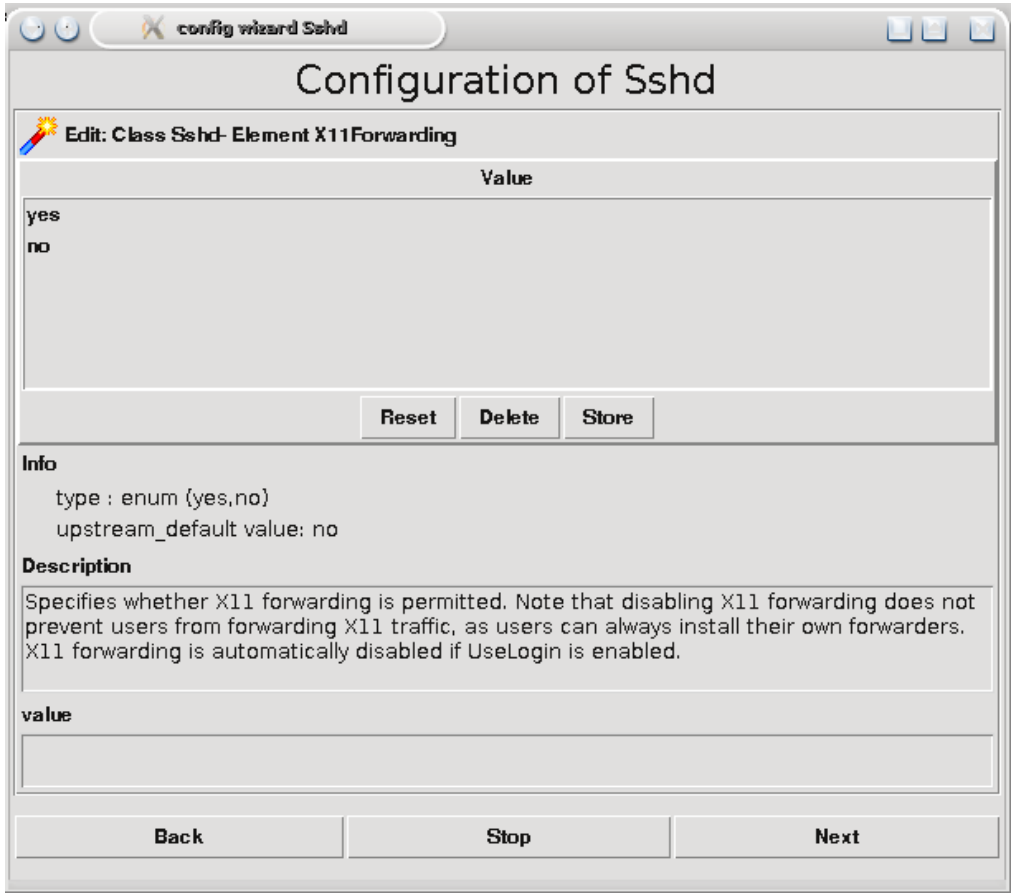
For more information on migration applied to software packages, see `http://wiki.debian.org/PackageConfigUpgrade`

Graphical interface

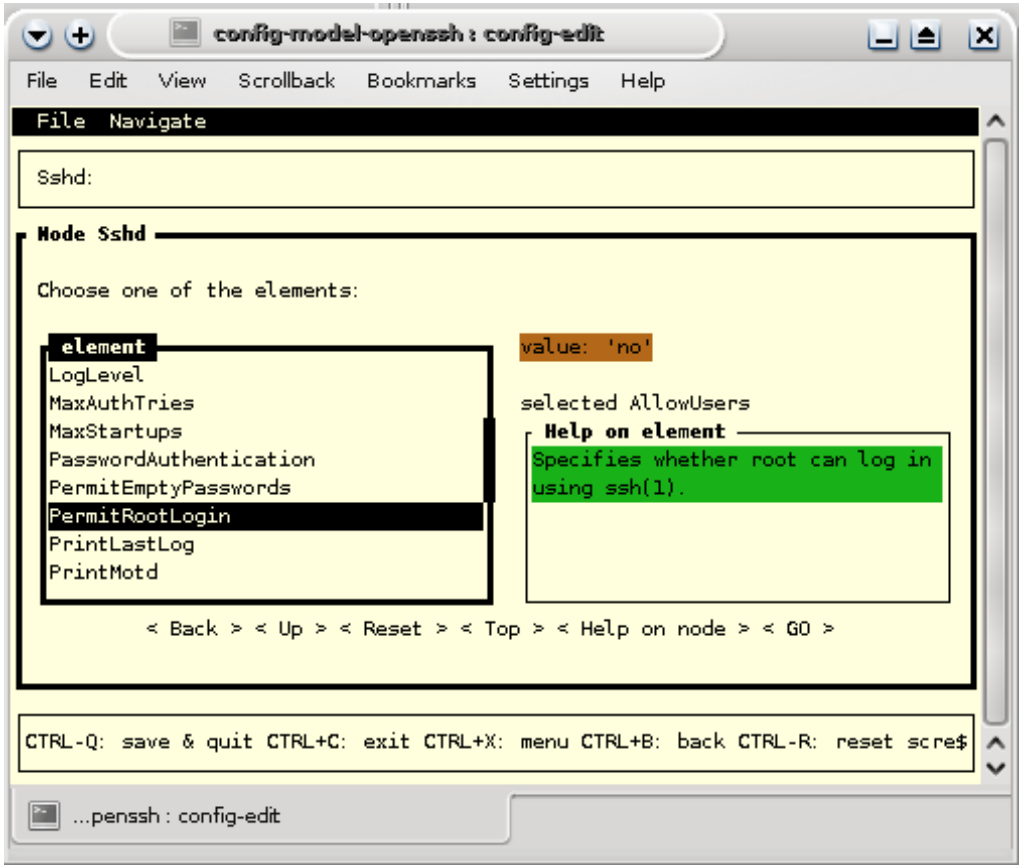


Note : Menu Option –> experience to show more parameters

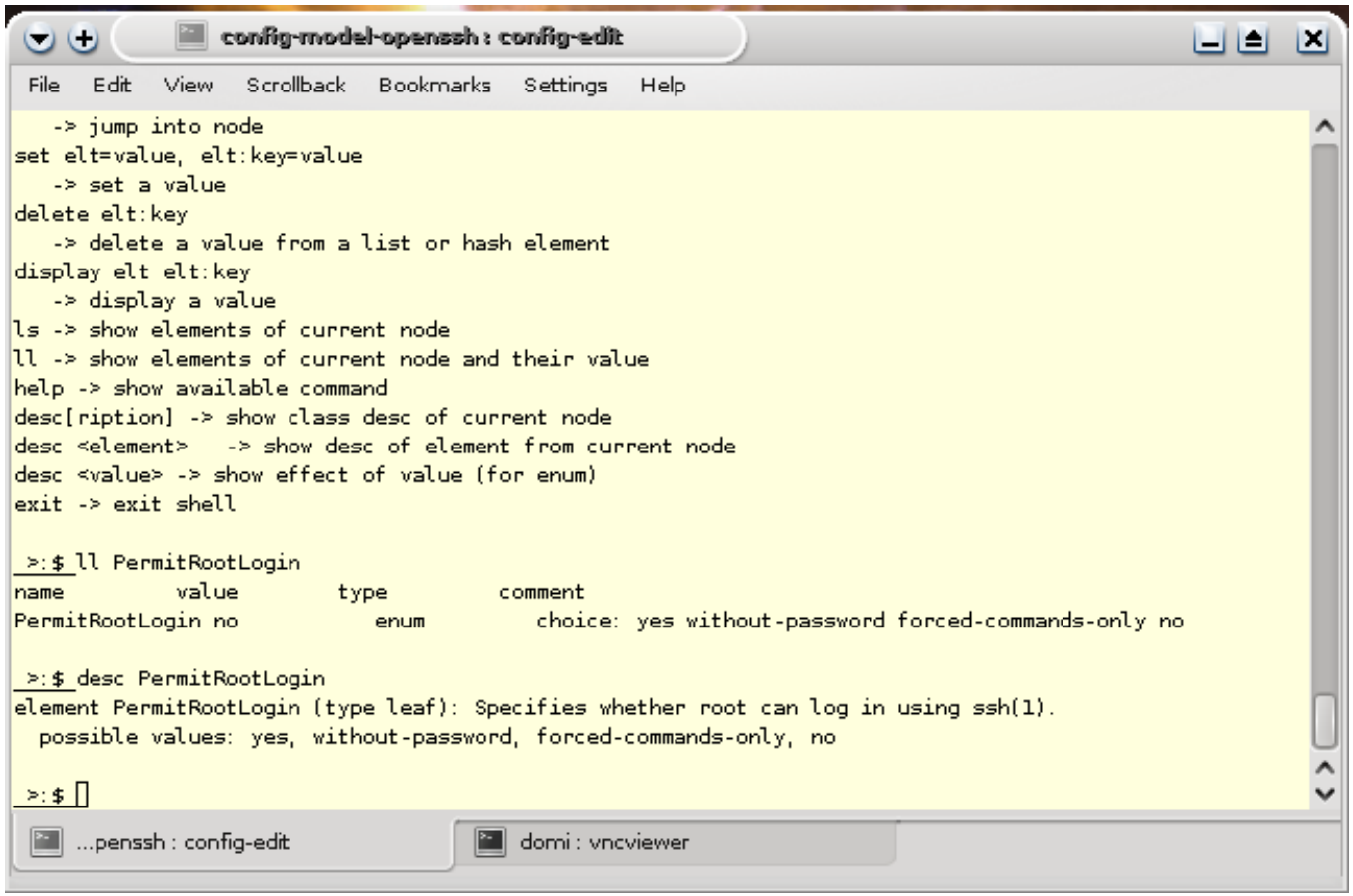
Wizard


config wizard Sshd
Configuration of Sshd
Edit: Class Sshd- Element X11Forwarding
Value
yes
no
Reset
Delete
Store
Info
type : enum (yes,no)
upstream_default value: no
Description
Specifies whether X11 forwarding is permitted. Note that disabling X11 forwarding does not prevent users from forwarding X11 traffic, as users can always install their own forwarders. X11 forwarding is automatically disabled if UseLogin is enabled.
value
Back
Stop
Next

Curse interface



Shell interface



Usage from programs

Command line :

```
$ sudo config-edit-sshd -ui none PermitRootLogin=no
2009/05/15 14:03:28 load model Config/Model/models/Sshd.pl
2009/05/15 14:03:28 Creating class Sshd
2009/05/15 14:03:29 Backing up file /etc/ssh/sshd_config
2009/05/15 14:03:29 writing config file /etc/ssh/sshd_config
```

Perl program :

```
$ sudo perl -MConfig::Model -e '
  my $i = Config::Model -> new -> instance(root_class_name=>"Sshd");
  $i->config_root->load("PermitRootLogin=no");
  $i->write_back;'
```

What is Augeas ?

Augeas is

- RedHat Emerging technology project
- Augeas is a configuration editing tool (API, command line).
- Parses configuration files and transforms them into a tree.
- Tree structure and file syntax is described in a lens

Lens

```
let key_re = /[A-Za-z0-9]+/ - /MACs|Match|AcceptEnv/
let other_entry =
    let value = store /[^ \t\n]+([ \t]+[^ \t\n]+)*/ in
    [ key key_re . sep . value . eol ]
```

What is Augeas ?

Augeas is

- RedHat Emerging technology project
- Augeas is a configuration editing tool (API, command line).
- Parses configuration files and transforms them into a tree.
- Tree structure and file syntax is described in a lens

Lens

```
let key_re = /[A-Za-z0-9]+/ - /MACs|Match|AcceptEnv/
let other_entry =
    let value = store /[^ \t\n]+([ \t]+[^ \t\n]+)*/ in
    [ key key_re . sep . value . eol ]
```

Augeas pros and cons

Pros

- Original structure and comments are kept
- Bidirectional transformation
- Common configuration API
- Available in several languages
- Extensible by writing new lenses

Cons

- Few semantic knowledge in Augeas lens
- Hard on users (no help, almost no validation)

For more information on Augeas, see
`http://et.redhat.com/page/Main_Page`

Augeas pros and cons

Pros

- Original structure and comments are kept
- Bidirectional transformation
- Common configuration API
- Available in several languages
- Extensible by writing new lenses

Cons

- Few semantic knowledge in Augeas lens
- Hard on users (no help, almost no validation)

For more information on Augeas, see
`http://et.redhat.com/page/Main_Page`

Using Augeas from Config : :Model

User gets best of both worlds

- Comments and structure are preserved by Augeas
- Usability and GUI provided by Config : :Model

Developer is not no lucky

- Must create model (for Config : :Model)
- Must create lens (for Augeas)
- Both tree must have similar structure
- Declare Augeas backend in model (can be tricky)

Using Augeas from Config : :Model

User gets best of both worlds

- Comments and structure are preserved by Augeas
- Usability and GUI provided by Config : :Model

Developer is not no lucky

- Must create model (for Config : :Model)
- Must create lens (for Augeas)
- Both tree must have similar structure
- Declare Augeas backend in model (can be tricky)

Configuration and package upgrades

Package upgrade :

- RedHat : Configuration evolutions leave rpm.new or rpm.save file
- Debian : Configuration evolution either :
  - trigger questions (often cryptic)
  - expose details to user with a diff
  - leave spurious files (dpkg-new or dpkg-old)

In all cases

Merging configuration requires good knowledge from user.

Configuration and package upgrades

Package upgrade :

- RedHat : Configuration evolutions leave rpm.new or rpm.save file
- Debian : Configuration evolution either :
  - trigger questions (often cryptic)
  - expose details to user with a diff
  - leave spurious files (dpkg-new or dpkg-old)

In all cases

Merging configuration requires good knowledge from user.

Configuration and package upgrades

Proposal

Use Config : :Model to merge :

- user data from config file
- package/upstream evolutions from model

Models with merge capability can be implemented by :

- Upstream projects
- Distributions (Debian, RedHat ...)
- Derived distribution (Knoppix, SkoleLinux ...)

Each can improve model coming from upstream

See proposal for Debian :

`http://wiki.debian.org/PackageConfigUpgrade`

Configuration and package upgrades

Proposal

Use Config : :Model to merge :

- user data from config file
- package/upstream evolutions from model

Models with merge capability can be implemented by :

- Upstream projects
- Distributions (Debian, RedHat ...)
- Derived distribution (Knoppix, SkoleLinux ...)

Each can improve model coming from upstream

See proposal for Debian :
`http://wiki.debian.org/PackageConfigUpgrade`

Configuration upgrade example

sshd_config : TCPKeepAlive option was formerly called KeepAlive.

```
KeepAlive => { value_type => 'enum',
               status     => 'deprecated',
               type       => 'leaf',
               choice     => [ 'no', 'yes' ]
             },
TCPKeepAlive => { value_type => 'enum',
                  status     => 'deprecated',
                  type       => 'leaf',
                  choice     => [ 'no', 'yes' ],
                  compute
                  => { formula   => '$keep_alive',
                       variables => { keep_alive => '- KeepAlive' },
                       allow_override => 1
                     },
```

Package upgrade howto

Debian

In postinst :

```
dh_config_model_upgrade --model_name Sshd \
--model_package libconfig-model-sshd-perl
```

RedHat

In postinst :

```
config-edit --model Sshd -ui none -save
```

Package upgrade howto

Debian

In postinst :

```
dh_config_model_upgrade --model_name Sshd \
--model_package libconfig-model-sshd-perl
```

RedHat

In postinst :

```
config-edit --model Sshd -ui none -save
```

Project status

Available Models

- OpenSsh
- Approx
- Config : :Model
- Krb5
- Xorg

Backend

- INI syntax
- Perl data structure
- YAML (on-going)
- Augeas

Community

- Debian packages
- Rpm packages (under constructions)
- Mandriva packages
- Proposal and patches for dh_config (package upgrades)
- Article in GNULinux Mag France

Project status

Available Models

- OpenSsh
- Approx
- Config : :Model
- Krb5
- Xorg

Backend

- INI syntax
- Perl data structure
- YAML (on-going)
- Augeas

Community

- Debian packages
- Rpm packages (under constructions)
- Mandriva packages
- Proposal and patches for dh_config (package upgrades)
- Article in GNULinux Mag France

Project status

Available Models

- OpenSsh
- Approx
- Config : :Model
- Krb5
- Xorg

Backend

- INI syntax
- Perl data structure
- YAML (on-going)
- Augeas

Community

- Debian packages
- Rpm packages (under constructions)
- Mandriva packages
- Proposal and patches for dh_config (package upgrades)
- Article in GNULinux Mag France

Future projects

Interfaces

- Wizard for the GUI (done)
- Search parameters and values
- Annotations

backend

- JSON
- XML

We need you !

Config : :Model needs your help :

- Integration in distros
- Multi-level configuration
- Plug-in mechanism for models (Xorg drivers)
- Define mechanism for configuration injection (e.g. mercurial viewer in Apache)

Future projects

Interfaces

- Wizard for the GUI (done)
- Search parameters and values
- Annotations

backend

- JSON
- XML

We need you !

Config : :Model needs your help :

- Integration in distros
- Multi-level configuration
- Plug-in mechanism for models (Xorg drivers)
- Define mechanism for configuration injection (e.g. mercurial viewer in Apache)

Future projects

Interfaces

- Wizard for the GUI (done)
- Search parameters and values
- Annotations

backend

- JSON
- XML

We need you !

Config : :Model needs your help :

- Integration in distros
- Multi-level configuration
- Plug-in mechanism for models (Xorg drivers)
- Define mechanism for configuration injection (e.g. mercurial viewer in Apache)

Links

- Config : :Model site
  `http://config-model.wiki.sourceforge.net`
- Config : :Model on CPAN
  `http://search.cpan.org/dist/Config-Model/`
- Config : :Model user mailing list `https://lists.sourceforge.net/lists/listinfo/config-model-users`
- GNU/Linux Mag France n°117 "Config : :Model - Créer un éditeur graphique de configuration avec Perl (1ère partie)"
- GNU/Linux Mag France n°120 "Config : :Model - Créer un éditeur graphique de configuration avec Perl (2e partie)"
- Proposal to use Config : :Model to upgrade configuration during Debian package upgrade
  `http://wiki.debian.org/PackageConfigUpgrade`
- Augeas project `http://augeas.net`