Jones CTP Scanner Reference Manual

Greg Baker (gregb@ifost.org.au)

Version 1.0.1

Contents

Chapter 1

Overview

`jones` is a fast, efficient and sophisticated premium scanner for underwriters. It fetches premium pricing for NSW Compulsory Third Party insurance from the NSW Motor Accidents Authority (MAA) price comparison website.

- It can collect every distinct premium for every class of vehicle across all answers to all rating variables in under half an hour. Sophisticated algorithms minimise unnecessary queries, and a parallel architecture runs searches concurrently.
- Output options including HTML, CSV, plain text and many, many other formats.
- Historical data can be kept without limit, and reports generated periodically.
- Quick-response searches can be run several times a day to alert you of any changes in your competitor's pricing – via email, SMS or RSS feed.

Chapter 2

Installation

2.1 Requirements

`jones` runs on any platform supported by Perl. This includes MS-Windows, Linux, Solaris, HP-UX, OpenVMS and many other platforms.

`jones` can be configured to use as little as 30-40MB of memory and a small percentage of a modern CPU, or it can be configured to run very fast and keep a large multi-cpu system fully occupied.

It does not even require administrator-level access to install and run. No GUI required, jones can run as a service or periodic scheduled job.

`jones` stores its historical data in many tiny flat files so it is best installed on a file system with a small block size, otherwise it will be somewhat wasteful of disk space.

2.2 MS-Windows

2.2.1 Perl

You will need to install a version of the Perl5 interpreter. This is *not* included in the `jones .EXE` and needs to be downloaded and installed separately.

ActiveState's Perl distribution has been well tested with `jones`, and can be downloaded for free from https://www.activestate.com/activeperl/downloads. Choose the latest version, and install it.

2.2.2 jones application

Double-click on the `jones .EXE` file to begin the installation. You will see a dialogue box as shown in figure 2.1.

Click on "Next". There is no choice screen offering an alternate directory to install to. `jones` expects to install into `C:\jones`, as shown in the dialogue box in figure 2.2

Figure 2.1: Installation dialogue launch

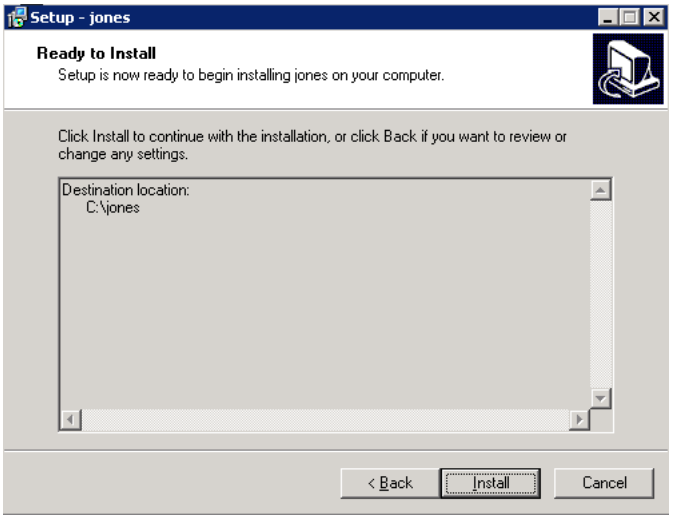

Figure 2.2: `jones` cannot install into any other directory

After the `jones` application files have been copied, the installer will ask whether to configure proxy access and scheduled jobs as shown in figure 2.3.

If you leave the "Configure proxy parameters in order to work with Microsoft ISA proxy service" ticked, notepad will be launched, where you should put your username, password, domain and proxy server address.

If your organisation uses no proxy, or uses another proxy server from any vendor other than Microsoft, untick this checkbox as `jones` will be automatically detect and use the correct proxy server to use from your environment without further configuration.

Figure 2.3: Dialogue for creating scheduled jobs, launching this document and configuring proxy settings

Each scheduled job ticked in the installer will launch a command-prompt session, which will ask for a password in order to register and run, as shown in figure 2.4.

2.2.3 Historical data and sequence numbers

You may also have received a zip file `jones-data.zip` containing some historical data.

- This is not needed if you are just using `jones` for fast alerting.
- If you don't have any pre-existing data, then the first time you run `jones` on each `.jns` file you will receive a warning. This can be safely ignored, and it will work normally on the second run.

Extract it to `C:`, so that it populates `C:\jones-data`. This may take a long time to run since it will usually have to reconstruct many tens of

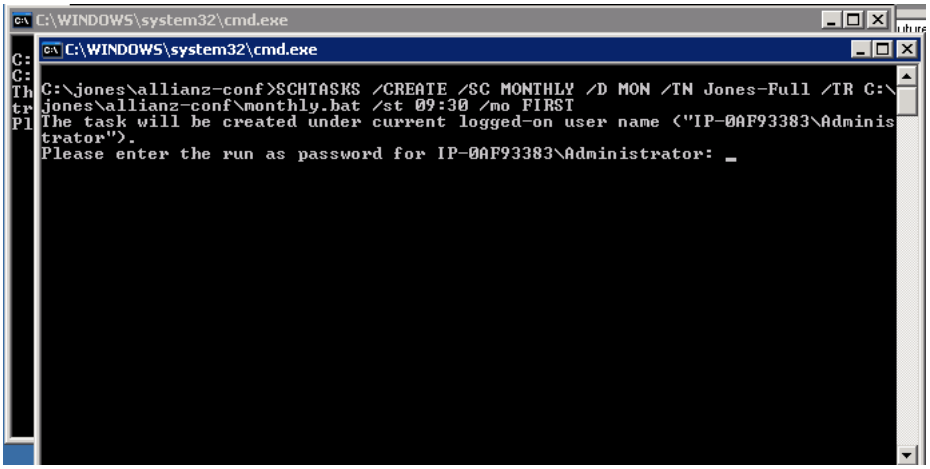

Figure 2.4: `schtasks` password prompt

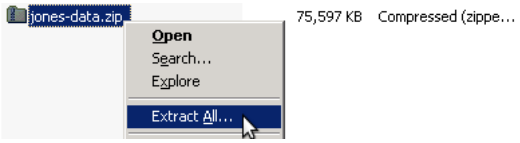

Figure 2.5: Navigate into the zip file, and click on “Extract all Files”

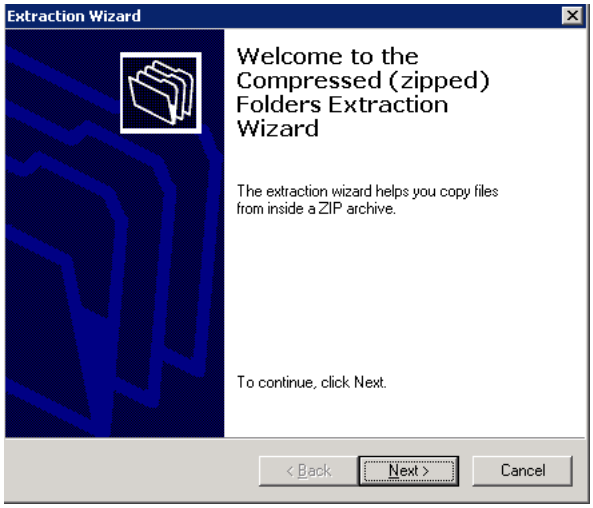

Figure 2.6: The Extract Wizard dialogue

thousands of files. The screenshots in figures 2.5, 2.6 and 2.7 show the steps, but unfortunately the MS-Windows shell extension for handling zip files is buggy and often misses files in large archives. If in doubt, try Info-ZIP's unzip utility from ftp://ftp.info-zip.org/pub/infozip/win32/unz600xn.exe

2.2.4 Changing the location

If you plan to run `jones` from the `C:` drive, skip this subsection.

`jones` expects to run from `C:\jones`, have its permanent database in `C:\jones-data`, and put its output into `C:\jones-output`. However, these can be links to folders on other drives.

As long as `jones` is not currently running a job, simply move the relevant folder onto another drive, and then create a junction from `C:` back to where you put it. Use the Sysinternals Junction tool (download from http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx) to create a junction. e.g. if you moved the data to the `D:` drive, run `junction C:\jones-data D:\jones-data`

A more advanced, alternative solution is to modify the `.jns` files in `C:\jones` to put the permanent database (and outputs) into other directories.

`jones` does not require file-level locking, and so these directories can be on networked-drives as well. This has not been tested well, and may perform badly due to the large number of network file reads and writes.

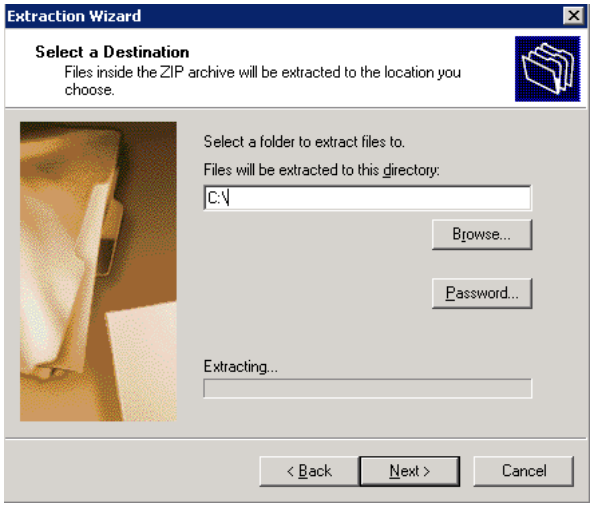

Figure 2.7: Change the extraction location to be `C:`

Chapter 3

Commands and Config

3.1 jones.pl

A program for keeping up with the changes made by CTP insurers

SYNOPSIS

`jones.pl` [`--progress`] [`--threads` *maxthreads*] [`--no-fetch`] [`--no-alerts`] [`--no-exports`] *config-file* [*config-file*...]

OPTIONS

–progress

Show a progress line to show that something is happening.

–threads *maxthreads*

Specify the limit of the number of threads (Windows) or child processes (Unix) to use to be *maxthreads*. This over-rides any configuration file parameter.

–no-fetch

Do not fetch from the MAA website, just use existing downloaded data. This is the same as specifying `enabled=no` in each configuration file.

–no-alerts

Do not report on changes which occured in this run compared to the last run even though the configuration file says to do so, even when there is a [changes] section.

–no-exports

Do not export data even when there is a section marked [export].

AUTHOR

(c) 2008,2009 The Institute for Open Systems Technologies Pty Ltd

3.2 jns

Configuration file format for *jones* CTP website scanner

OVERVIEW

A configuration file for *jones* must have a `[scan]` section and a `[storage]` section. It can optionally have a `[fetch]` section. In order to do anything useful, it will need an output in either the `[changes]` or `[export]` sections, which will probably be more useful if a `[historical]` section is defined.

These configuration files typically have a filename ending in `.jns`.

[scenarios]

The scenarios section defines what scenarios to scrape from the MAA website. Each line item can contain several values, separated by commas. *jones* will use every possible combination of the scenarios listed.

Several parameters (marked with *) can take a range (two values separated by a –) instead. When this is done, *jones* does smart searching, looking at the two extremes of the ranges and working its way between them skipping scrapes wherever possible.

Zone

Possible values are `Country`, `Metro`, `Newcastle`, `Wollongong` and `Outer`

History

Possible values are `Good Driver`,`Bad Driver`, `Good Driver and NRMA Member` and `Bad Driver without Roadside Assistance`.

Setting the last two of these makes no sense if Roadside is set as well, and means that `%r` is undefined as well.

Ownership

Possible values are `Privately Owned` and `Company Owned`

Usage

Possible values are `Private Usage` and `Business Usage`

Existing

Possible values are `CTP with Zurich`, `CTP with Allianz`, `No CTP` and `Lapsed CTP`.

Comprehensive

Possible values are `Comprehensive with AAMI`, `Comprehensive with Allianz`, `Comprehensive with CIC-Allianz`, `Comprehensive with GIO`, `Comprehensive with NRMA`, `Comprehensive with QBE`, `Comprehensive with Zurich`, and `No comprehensive`.

Gender

Possible values are `Male` and `Female`.

Roadside

Possible values are `No roadside assistance` and `NRMA Member`.

Vehicle

There are many, many possibilities here. e.g. `Toyota Prius`, `Ford Falcon XR8 Pursuit`. MCB class motorbikes (with engines larger than 300cc) aren't handled very well by *jones*.

VehicleAge *

This can be specified as by the year of manufacture or as the number of years old.

DriverAge *

Valid values are between 16 and 100 inclusive.

Commencement *

Valid values are between 0 and 60 inclusive. This represents the number of days in the future when the policy is to commence.

[fetch]

This section defines parameters *jones* about the way the fetch occurs

threads

How many processes (on Unix) or threads (on Windows) to run concurrently. The default is 32, and can be over-ridden on the command-line with the `--threads` option. The thread is very coarse – each discrete scenario gets its own thread, up to the threading limit defined here. This means that for a fetch which has a lot of ranges and very few discrete variables the number of threads used may be lower than specified here.

enabled

If set to a false value (e.g. "0", "no", "disabled", "off" or "false"), then no fetch will be performed, and pre-existing data will be used. Can be set on the command-line for all config files with `--no-fetch`. Defaults to being enabled.

[changes]

After the MAA website fetch has been completed for all scenarios, *jones* then checks to see if any premiums have shifted. The base-line is the fetch which was run closest to the "0" time from the `[historical]` section. This defaults to the previous fetch.

The outputs can be any combination of generating a file, sending an SMS or sending an email. If a file is being generated, then a command can be run as well.

Format

What to put on each line of the output for each change. Optional, but without it no file is produced.

Header

What to put as the first line in the report of changes. Optional.

Nothing

What to output to the file if there were no changes. Optional – if this is not set, then nothing is created or run when there are no changes.

Outfile

File to append the output from a change report onto. If this is missing, then output is printed out to STDOUT.

Time-based substitutions can appear in this path.

Command

Command to run after the `Outfile` has been generated. Optional. Note that the command will *not* be run if no changes were identified and `Nothing` is not set.

Time-based substitutions can appear in this path.

SMSusername

The username to use when connecting to the ValueSMS website in order to send an SMS alert. You receive this when you sign up at their website (`http://www.valuesms.com/`). Optional, but required if you want to send an SMS.

SMSpassword

The password to use when connecting to the ValueSMS website. Optional, but required if you want to send an SMS.

SMSphones

The list of phone numbers to send the message to, separated by commas. Optional, but required if you want to send an SMS.

SMShead

Similar to `Header`, but used for SMS messages. Optional, but required if you want to send an SMS.

SMStexts

Similar to `Format`, but used for SMS messages. Duplicate output texts are suppressed and there are no newlines between outputs so `SMStexts=%I;` would produce a semicolon-separated list of insurers who have made premium changes. Optional, but required if you want to send an SMS.

SMTPserver

The mail server's address.

For sites running MS-Exchange or Lotus Notes, contact your mail administrator and for this information since it may not be the same as the address used in your desktop email tool Your mail administrator may also need to enable SMTP mail reception from the computer you are running `jones` on.

Optional, but required if you want an email to be sent.

FromAddress

What address you want the emails to appear to come from. Optional, but required if you want an email to be sent.

ToAddress

A comma-separated list of addresses you want the emails to be sent to. Optional, but required if you want an email to be sent.

EmailSubject

The subject line of the email. Time based subsitutions can be used here. Optional, but required if you want an email to be sent.

EmailLines

Sames as Format, but for the body of the email message. Optional, but required if you want an email to be sent.

[export]

Outfile

The filename to write to. Time-based substitutions can appear in this path. Defaults to STDOUT.

Header

What to put on the first line of output data. Optional.

Format

What to put on each line of output data. If not present, no export is done.

Footer

What to put on the last line of output data. Optional.

Order

The sorting order. This is a list of columns, the first column is sorted first.

Command

A command to run after the export is completed.

The `Header`, `Format` and `Footer` fields can also have a number after them. e.g. `Header4` or `Format7`. This allows for (for example) historical exports in chunks. `jones.pl` will perform a complete export using the number-less `Header`, `Format` and `Footer` fields, and then work its way up through any other numbers present performing a complete export each time according to the `Header#`, `Format#` and `Footer#` parameters for that number.

[storage]

This section defines where to store the fetched data, and in what format.

Path

This specifies where to store the data which is collected. It can be a full path or relative path name, and will almost certainly have subsitution codes in it.

A typical unix example is:

```
/var/db/jones/city-worst/%I/%o-%u-%e-%t.txt
```

A typical Windows example is

```
C:\Program Files\Jones\Database\City Worst\%I\%o-%u-%e-%t.txt
```

See the `SUBSTITUTION VARIABLES` section for details on % sign substitutions.

In order to save data correctly, all parameters given in the `[scenarios]` section of the configuration file must be part of the path name, unless they don't take multiple values or are ranges. The insurer name should be part of the path too (`%I`)

Log

This tells *jones* where it can store a log timestamp to record the date and time of the last run. This does **not** get substituted with substitution variables.

[historical]

The `[historical]` section is only relevant if `Log` is set in the `[storage]` section, and affects Format and Header in the `[export]` and `[changes]` sections.

You can specify ten different moments in the past which you would like to compare against. Typically these would be times like: the previous invocation of the program; this time last week; this time last month; three months ago.

Of course, *jones* might not have run exactly one month ago (perhaps it was a weekend, or there was a problem with the MAA website that day). So *jones* will look through the `Log` file and find the fetch which occured as close as possible to the requested time.

Note that time moment 0 has a special meaning – this is the baseline against which changes are reported. A common value for this (which is the default) is 'previous'.

The syntax for specifying times is quite flexible, as can be seen in the following examples.

1. =previous
2. =1 month ago
3. =6 months ago
4. =1 year
5. =23rd June 2009
6. =-1 year + 3 days
7. =yesterday
8. =1 business day ago

Note that "1 month" is not the same as "1 month ago", and that business day calculations are a little flakey.

SUBSTITUTION VARIABLES

Scenario-based substitution variables

Some variables can be used as ranges, as discussed in the `[scenarios]` section, and are marked with *. These variables cannot be used as part of the `Path` (from the `[storage]` section of the configuration file) when they are used as ranges.

%z

Zone

%h

History

%o

Ownership

%u

Usage

%e

Existing

%c

Comprehensive

%g

Gender

%r

Roadside

%v *

The whole vehicle name including the string "%a years old".

%k

The vehicle name without reference to its age

%m *

The year of manufacture of the vehicle

%a *

The number of years old the vehicle is.

%d *

The driver's age.

%n *

The number of days in the future the policy will commence.

%i

A unique identifier number for this scenario. At the moment, this can only be used in [export] and [changes] sections.

%j

A command line to run which will print out the MAA website data.

Time-based and insurer-based substitution variables

Time-based variables can be used everywhere. They are based on the time that *jones* was launched not the time when a particular fetch has completed.

The insurer name can be used in filenames and line formats (but not headers).

There are historical variations available for all time-based variables.

%I

The insurer name

%Y

The two-digit year when the program was started

%X

The century when the program was started

%D

The day of the month when the program was started

%M

The number of the month when the program was started

%B

The abbreviated month name in English of the month when the program was started

%H

The hour when the program was started

%T

The number of minutes past the hour when the program was started

%R

The day of the month of the policy commencement.

%U

The month name when the policy is to commence.

%V

The century when the policy is to commence.

%W

The year when the policy is to commence.

Premium value variables

The simple premium value variables are only valid for `Format` lines in the `[output]` or `[changes]` section (not `Header` or `Nothing` lines).

All premium variables have historical variants.

%A

AAMI premium

%L

Allianz premium

%C

CIC-Allianz premium

%G

GIO premium

%N

NRMA premium

%Q

QBE premium

%Z

Zurich premium

%P

Current premium

Historical Variants

Time-based and premium-based variables can have a number inserted into them to refer to the time or premium which was found on a different invocation of the program.

For example, if there is a line `0=previous` in the `[historical]` section, then `%0P` means "the premium which was seen the last time the program was run".

As another example, if there is a line `5=1 year ago` in the `[historical]` section, then `%5Q` means "the QBE premium from this time last year".

SEE ALSO

jones.pl(1)

AUTHOR

(c) 2008,2009 The Institute for Open Systems Technologies Pty Ltd

Chapter 4

Programming Reference

4.1 Car Types

This package gives exhaustive lists of makes and models.

I copied-and-pasted %vehicle_database from the MAA website source. I guess a better (later) version of this program would read them from the MAA website too.

Really cool would be to pull apart VIN numbers as well. Maybe &guess() should recognise VIN codes and then translate.

(And handle the case when the RTA has put the wrong VIN code down for a car.)

vehicle_makes_in_vehicle_class

Given a vehicle class (1, 3c, 10a, 10b), return the three letter codes for all the makers of cars in that class. Defaults to class 1.

vendor_name_of_vendor_code

get_vendor_code_from_vendor_name

Optionally takes a vehicle_class in order to differentiate HOG from HOL.

car_models_from_vendor_code

Returns a hash reference. They keys are model names. The values are the model codes.

sanity_check_class_model_make

Expects a vehicle class, a make (three letter vendor code) and possibly a model.

guess

Takes a string as an argument. It tries to figure out what it is, and returns a list of:

vehicle year,

vehicle class

vehicle make (full name)

vehicle make (short code)

vehicle model (name)

vehicle model (short code)

If given a second argument, that is taken to be a vehicle class (i.e. 1, 3, 10a or 10b.)

At the moment it doesn't handle motorcycles very well.

vehicle_classes

Return a list (1,3,10a,10b) – the different classes of vehicle according to the MAA.

4.2 Combinatorix

This package implements some basic give-me-all-varations-on-this-theme functions.

pair_variations

`pair_variations($a1,$b1,$a2,$b2,$a3,$b3...)` returns a list of list references. The list will look like this:

$a1,$a2,$a3

$a1,$a2,$b3

$a1,$b2,$a3

$a1,$b2,$b3

$b1,$a2,$a3

$b1,$a2,$b3

$b1,$b2,$a3

$b1,$b2,$b3

variations_of_lists

`variations_of_lists($listref1,$listref2,$listref3...)` returns a list of list references. Each of these referenced lists will have one element from listref1, one from listref2, one from listref3 and so on.

variations

`variations($hashref` takes a hash reference as an argument. The values of `$hashref` should be list references. `variations($hashref)` returns a list of hash references. Each of these referenced hashes will have the same keys as the original `$hashref`, but the values will be scalars, in all possible combinations.
For example

```
%hash = ( 'dwarf' => [ 'Sleepy', 'Grumpy', 'Happy' ],
          'princess' => [ 'Snow White', 'Cinderella' ] )
@answer = variations(\%hash);
```

has the same effect as

```
@answer = ( {'dwarf' => 'Sleepy', 'princess' => 'Snow White' },
            {'dwarf' => 'Grumpy', 'princess' => 'Snow White' },
            {'dwarf' => 'Happy',  'princess' => 'Snow White' },
            {'dwarf' => 'Sleepy', 'princess' => 'Cinderella' },
            {'dwarf' => 'Happy',  'princess' => 'Cinderella' },
            {'dwarf' => 'Grumpy', 'princess' => 'Cinderella' } );
```

4.3 CTPPostcodes.pm

This module defines `@zones` and `get_zone`, based around `http://www.maa.nsw.gov.au/default.asp`

4.4 DateTags

Sometimes commencement date is a range, sometimes it is a discrete variable. This module wraps the difference up and hides it away.

commencement_field

`commencement_field($cfg)` returns the Commencement field from the Config::IniFiles object `$cfg`.

commencement_list

`commencement_list($cfgline)` returns the list of all ages specified in `$cfgline`, which might be quite a lot if $cfgline is a range.

commencement_earliest_and_latest

`commencement_earliest_and_latest($line)` return the minmum and maximum ages defined by the config line `$line`.

tags_list

`tags_list($cfg[,$expand_ranges])` returns a list reference which should be inserted into combinatorix input representing the discrete alternatives for the driver age.

This might be a long list (if Commencement is discrete) or a 1-element list "CommencementIn%nDays"

insert_range_fields

`insert_range_fields($cfg,$range_fields)` takes the Commencement field from $cfg and puts the minimum and maximum values into the hash reference $range_fields (and sets the subst field too).

If Commencement is discrete (and not a range), then this function will do nothing.

fudge_commencement

`fudge_commencement($hashref)` will look to see if it can figure out what the driver age is from a hash of scenario parameters, and push it back into the hash reference.

4.5 DriverTags

Sometimes driver age is a range, sometimes it is a discrete variable. This module wraps the difference up and hides it away.

age_field

`age_field($cfg)` returns the DriverAge or "Driver Age" field from the Config::IniFiles object `$cfg`.

age_list

`age_list($cfgline)` returns the list of all ages specified in `$cfgline`, which might be quite a lot if $cfgline is a range.

age_min_max

`age_min_max($line)` return the minmum and maximum ages defined by the config line `$line`.

tags_list

`tags_list($cfg[,$expand_ranges])` returns a list reference which should be inserted into combinatorix input representing the discrete alternatives for the driver age.

This might be a long list (if DriverAge is discrete or $expand_ranges is something true) or a 1-element list "DriverAge%d"

insert_range_fields

`insert_range_fields($cfg,$range_fields)` takes the DriverAge field from $cfg and puts the minimum and maximum values into the hash reference $range_fields (and sets the subst field too).

If DriverAge is discrete (and not a range), then this function will do nothing.

fudge_driver_age

`fudge_driver_age($hashref)` will look to see if it can figure out what the driver age is from a hash of scenario parameters, and push it back into the hash reference.

4.6 ExportsConfig

The parameters in the [exports] section of a jones configuration file are:

Header

Footer

Format

Order

Outfile

Command

The first three of these can also have a number after them, e.g. Header4. Numbered parameters are processed after unnumbered ones and in increasing order.

i.e. A rather pointless [exports] section could look like this:

```
[exports]
Header=Hello
Format=World
Header1=Goodbye
Format3=Planet
Header4=I'm back
Footer=Farewell
```

If there are only two scenarios defined, the output from this would be:

```
Hello
World
World
Goodbye
Planet
Planet
I'm back
Farewall
```

The `Order` parameter sorts the output. `Outfile` can be a filename (date and time-based substitutions are available) or "-" to mean STDOUT.

new

`new($class,$cfg)` takes a Config::IniFiles object and reads the section marked [exports]

about

`about($xpt,$topic)` returns a hash based on the ExportsConfig object's Config::IniFiles object. If the config file has:

```
[export]
Header=This is a header
Header3=This is a third header
Header7=This is a seventh header
Footer=This is the end
```

Then `about($xpt,"Header")` will return

```
{ Header => 'This is a header',
  Header3 => 'This is a third header',
  Header7 => 'This is a seventh header'
}
```

It still attempts to return valid data even if `$xpt` is disabled.

sequences

`sequences($xpt)` returns a list, each element can be used as an argument to `get`. In the example above (from `about(...)`) it would return the list `("",3,7)`.

get

`get($xpt,$what,$sequence_number)` gets the relevant parameter from the [export] section of the Config::IniFiles object.

`$what` can be one of "Format", "Header" or "Footer".

The sequence number is either an empty string or a number.

includes_historical

`includes_historical($xpt)` returns true is there is any historical data defined to be exported.

disable

`disable($xpt)` turns off exports, and makes sure almost all functions return "nothing" values.

enabled

`enabled($xpt)` returns true if `$xpt`->disable(...)> or its equivalent has not been called.

`check_timelog($xpt,$storage_paths)` will confirm that there is a 'Log' parameter defined in the [storage] section of the config file, and will disable exports if the Log points to a non-existent file.

outfile_template

`outfile_template($xpt)` returns the 'Outfile' parameter from the [export] section, or "-" if none was defined.

ordering

`ordering($xpt)` returns the 'Order' parameter from [export] and returns a list reference.

command

`command($xpt)` returns the 'Command' parameter from [export] or undef otherwise.

4.7 FetchScenario

This package calls MAACTP with appropriate tags for the kind of fetch operation required.

fetch

`fetch($variations_ref,$range_fields,$max_threads,$storage_paths,$show_progress)` gets passed:

reference to a list of list references

Typically this would be generated by a call to `Combinatorix::variations`

reference to a hash of ranges

Ranges have a keys which are an axis names, and values which are themselves hash references. These hash references have keys 'low', 'high', 'subst' and (optionally) 'year-minus-subst'.

the maximum number of threads (or unix processes) to spawn

the template of where to save the data

whether or not to show progress statements

spawn

`spawn($max_threads,$variation,$scenario_count,$child_procs)` checks to see if `$max_threads` is greater than 1, because if it isn't, there's no spawning to do – all processing happens in the parent.

`$variation` and `$scenario_count` are stored into the `$child_procs` hash reference for debugging purposes later.

If returns 0 if this process is supposed to do some work, or a PID if we are a lazy parent.

end_spawn

`end_spawn($max_threads)` is the counterpart to `spawn(...)`. If there was any spawning done, then assume that we are a worker thread and exit.

reap_afterwards

`reap_afterwards($child_procs,$show_progress)` repeatedly does `wait()` calls, removing the dead PID from the `$child_procs` hash. If `$show_progress` is something true, it prints out an informative message show what it is up to.

fetch_ranges

`fetch_ranges()` **IS NOT DOCUMENTED PROPERLY YET!**

simple_fetch

If a scenario doesn't have any ranges, then it's really easy.

`simple_fetch($scenario,$storage_paths)` fetches from MAACTP website the scenario defined by the hash ref `$scenario` and saves it into the file specified by `$storage_paths`.

read_all_insurer_history

`read_all_insurer_history($cache,$scenario,$storage_paths,@time_history)` inserts into `$scenario` all the insurer premiums for the scenario defined in `$scenario` for the present time plus all the times listed in `@time_history` as stored in the databases of `$storage_paths`

It also calls the various fudge_* functions to fill out `$scenario`, which has the side effect of calling MAACTP::reset().

read_premium_from_file

`read_premium_from_file($cache,$scenario,$storage_paths,$insurer,[$time])` reads a single file, as written by `simple_fetch(...)` or `fetch_ranges(...)` and returns the premium.

The optional argument `$time` can be a hash reference to a different timeset, in the format requried by `JonesConfig::substitute()`

alert_changes

`alert_changes($scenarios_listref,$timestamp_history,$storage_paths,$headline` compares the current and previous fetch runs for the scenarios of `$scenarios_listref` and returns a list describing the differences.

It uses the listref `$timestamp_history`) to figure out what moment in time is "previous", and `$storage_paths` to get the data for this run and the last one.

If there are no differences, then the returned list will have `$no_change_string` as its only element, or an empty list if `$no_change_string` is `undef`.

If there are differences, then the returned list will be based on the strings `$headline` and `$eachline`. The first element of the list will be `$headline` (with any `%` substitutions performed). The remaining elements will be based around `$eachline`. There will be a list element for each difference discovered between the current and previous fetch runs.

export_scenarios

`export_scenarios($scenarios_listref,$time_history,$storage_paths,$line_format)` steps through every scenario in `$scenariso_listref` and reads the data stored in the appropriate translation of `$storage_paths` (for all the times of `$time_history`), and prints it out in the format specified by `$line_format`.

4.8 Hypercorners

This package creates strings which can be used as unique identifiers for the corners of an N-dimensional hypercube.

corners

`corners(N)` returns a list of 2**N identifiers. The list is cached, so it's reasonably efficient to call it many times.

first_corner

`first_corner(N)` creates a string which will be used as a key for a corner. It consists of *N* comma-separated zeroes. It can be fed into the `next_corner()` function later.

You probably don't need to call this. Call `corners(N)` instead.

next_corner

Returns a string of the same length as its argument, or undef if the strings was a sequence of comma-separated ones.

You probably don't need to call this. Call `corners(N)` instead.

set_axis_N

`set_axis_N($corner_index,$axis_number,$value` sets the $axis_number'th element in the corner_index to $value (which must be 0 or 1). axes start at axis number 0.

get_axis_N

`get_axis_N($corner_index,$axis_number` returns the Nth element of the corner_index vector.

matching

`matching($pattern)` returns the list of corners which match $pattern. $pattern is a string consisting of comma-separated 1,0 or *.

4.9 Hypercube

This package implements a class which models a hypercube where each vertex may have an associated value.

It can report which vertexes are missing values and whether all values are the same along a given axis.

new

`new Hypercube(N)` constructs an unvalued N-dimensional hypercube.

usual_value

`usual_value($hypercube)` only works on incomplete or constant hypercubes. If the last argument to `set()` has always been the same, it returns that, otherwise it returns undef.

complete

`complete()` returns 1 if all corners have values, and 0 otherwise.

set

`set($hypercube,$corner_idx,$new_value)` sets the value on a corner.

set_list

`set_list($hypercube,$value,$corner_idx1,$corner_idx2,...)` repeatedly calls `$hypercube`–set($corner_idxX,$value) for each corner index.

get

`get($hypercube,$corner_index)` returns the value (if any) associated with the corner `$corner_index`.

keeps_constancy

`keeps_constancy($hypercube,$value)` returns 1 if we could call `set($hypercube,_,$value)` and `constant($hypercube)` would still return a defined value. 0 otherwise.

constant

`constant($hypercube)` returns the value if all corners are defined and have the same value, undef otherwise.

constant_sanity_check

`constant_sanity_check($hypercube)` returns the value if all corners are defined and have the same value, undef otherwise.

missing

`missing($hypercube)` returns the first corner which doesn't have an associated value defined yet.

inconstant_along_axis

`inconstant_along_axis($hypercube,$N)` returns 1 if we are sure there it is impossible for axis N to be constant. If we are not sure (e.g. because one or other end of something in axis N is undef still) we return 0.

first_inconstant_axis

`first_inconstant_axis($hypercube)` returns the lowest axis_number which returns 1 from `inconstant_along_axis($hypercube,$x)`. Returns undef if there aren't any.

mitosis

`mitosis($hypercube,$axis,[$flatten_first,$flatten_second])` returns two hypercubes formed from the original, as if the original were cut along `$axis`.

If the optional arguments `$flatten_first` or `$flatten_second` is nonzero, then the respective hypercube is created as if `$axis` was compressed.

For example, `$hypercube` is 2-dimensional and has values

```
(0,0) = 5       (0,1) = 4
(1,0) = 6       (1,1) = 7
```

Then `$hypercube->mitosis(0)` will create two hypercubes. The first one will be

```
(0,0) = 5       (0,1) = 4
(1,0) = undef   (1,1) = undef
```

and the second one will be

```
(0,0) = undef   (0,1) = undef
(1,0) = 6       (1,1) = 7
```

`$hypercube-`mitosis(0,0,1)> on the ther hand would create two hypercubes like this:

```
(0,0) = 5        (0,1) = 4
(1,0) = undef    (1,1) = undef
```

and

```
(0,0) = 6        (0,1) = 7
(1,0) = 6        (1,1) = 7
```

dump

`dump($hypercube)` returns a string which shows the hypercubes vertex values.

4.10 IsaCompat

Microsoft in their unusual wisdom decided to create a proxy service for accessing the internet which only works properly if you use a special Microsoft-developed authentication protocol.

No Perl library for this yet exists properly, so we have to launch a special local daemon to act as a proxy, which then connects to the Microsoft proxy, supplying appropriate credentials.

using_isa

Returns true if ISA proxy compatibility has been turned on by creating a file called `proxy.cfg` in the same directory as the program.

proxy_url

Return a list suitable for passing to LWP::UserAgent::proxy which connects to the proxy which was spawned.

launch_cntlm

Starts up the cntlm daemon if it is required and is not already running.

kill_cntlm

Kills off the cntlm daemon.

4.11 JonesConfig

The following functions implement the behaviour documented mostly in *../doc/jns.pm*.

Constants

MAX_HISTORY

How many different elements are read in the [historical] section.

substitute

`substitute($variables,$str,[@time_hashes])` substitutes the percent-string placeholders from jns in *doc* with their values from `$variables` into `$str` and returns the result.

save_time

`save_time($filename)` appends the program start time into `$filename` in a format which can be easily editted, and which can be read by `load_time_of_most_recent`.

load_time_of_most_recent

`load_time_of_most_recent($filename)` reads the last few lines from `$filename` and returns a hash compatible with the third argument of `substitute`.

load_time_closest_to

`load_time_closest_to($filename,$goal_time)` reads through `$filename` and finds the entry there which is closest in time to `$goal_time`.

`$goal_time` should be in the same format as `save_time()` produces, i.e. `%D/%M/%X%Y %H:%T:%S`

read_scenarios_section

`read_scenarios_section($cfg[,$expand_ranges])` reads the [scenario] section from the `Config::Inifiles` object `$cfg` and returns two hash arrays (the discrete elements and the range-based elements).

If the optional argument `$expand_ranges` evaluates to something true, then there will be no range-based elements. This will make the discrete elements array very much larger.

fudge_command_line

`fudge_command_line($scenario)` inserts back into the `$scenario` an extra key-value pair command_line="..." which contains the command line arguments needed to reproduce this scenario.

This calls `MAACTP::reset()` so should only be used when you are not about to launch a fetch!

4.12 Motor Accidents Authority CTP Website Access Library

This package queries the NSW Motor Accdient's Authority website to see what prices various insurers are charging for compulsory third party (CTP greenslip) insurance.

getopt_options

`getopt_options()` returns a list suitable for passing to `Getopt::Long::GetOptions` so that you can extract all the appropriate information to call `get_pricing()` from the command-line.

errors()

`errors()` looks through the data in `@questions` and returns an empty list if everything is fine (which is a bit odd, I know). If there is missing data, or incorrect or ambiguous data, it returns a list of error messages about the problems.

cmd_line

`cmd_line()` returns a list of command-line arguments which would reproduce the current state.

fetch_pricing_page

`fetch_pricing_page($pricing_parameters)` performs the HTTP POST operating to the MAA web page using the parameters specified in `$pricing_parameters` which is a hash of CGI key/value pairs of the answers to the questions on the form from http://prices.maa.nsw.gov.au/.

make_pricing_parameters

`make_pricing_parameters()` returns a hash reference of CGI pairs formed form the data in `@questions`.

get_premium

`get_pricing([$parameters])` returns a hash reference, based around the data in `@questions`, or based around `$parameters` if that is set.

The $answers hash should have a value for each key in (sort values %MAACTP::questions).

The returned hash will have insurer names as keys, and the values being the prices they would charge to insure the vehicle.

parse_maa_webpage

`parse_maa_webpage($raw_source)` parses the raw HTML that you get back from the MAA website.

4.13 NSWPostcodes

This package was mostly created by grabbing the Auspost postcodes data file and running it through

```
cut -d',' -f1,2 | sed -e 's/,/ = /' -e 's/$/,/'>
```

It provides two associative arrays:

%codes, which is keyed from a postcode with values being a reference to a list of all the suburb names with that postcode.

%names which is key from a suburb name and returns all postcodes used in that suburb.

4.14 PremiumTable

A premium table object has a RangeDB for each insurer.

In the future I'll subclass this somehow (or something like that) and cope with tables which have no ranges in them.

insurers

`insurers()` returns a list of the names of NSW CTP insurers.

new

`new PremiumTable($base_scenario,$range_fields)` creates a new PremiumTable object.

`$base_scenario` should be a reference to a hash which has MAACTP tags as values.

`$range_fields` should be a reference to a hash which has key which are the names of the axis ranges for the internal RangeDBs. The values of `$range_fields` should be hash references with keys 'low', 'high', 'subst' and (optionally) 'year-minus-subst'.

next_needed_fetch

`next_needed_fetch($table)` returns an object which can be passed to `tags_of_fetch` and `store`, which represents the next pricing scan we will need to do.

tags_of_fetch

`tags_of_fetch($table,$handle)` returns a tag list suitable for feeding to `MAACTP::use_tags()`.

store

`store($table,$handle,$premiums_hash)` saves the premiums from each insurer (stored in `$premiums_hash`) into its internal RangeDB objects and associates them with `$handle`.

save

`save($table,$path_template)` saves its RangeDB objects into the file specified by `$path_template` after the usual *JonesConfig* substitutions have been made.

4.15 RangeDB

This package implements an odd kind of database.

Suppose you have a function of many variables F(a,b,c,...,z), and you also know that for any variable V, dF/dV = 0 only once for fixed values of the other variables, or otherwise dF/dV = 0 for all V.

That is, if you took a slice of F with nearly every variable fixed, and graphed the differing values F took as you varied just one variable, the picture would be one of:

a monotonic function

a parabola

a half sinusoid

a hill

a valley

Now suppose it was quite expensive to compute F(a,b,c,...z). What you can do is take a few sample points and see if there are any equi-valued planes or lines through the data. If there are, then you don't need to calculate every value along such a line – you can assume that they are equal just based on the two points you've got so far. Later you can come back and check to see if you might have stripped the top off a peak, or filled the bottom of a valley.

This package implements the kind of database structure you need to hold this kind of data.

new

`new()` creates a new database instance. It takes as an argument the name of each axis.

e.g.

```
new RangeDB('age','weight','height');
```

splinter

`splinter RangeDB ($parent,$split_axis,$replaced_lower,$replaced_higher,$hypercube)` is a constructor

set_axis_range

`set_axis_range($database,$axis_name,$low_value,$high_value)` is one of the essential functions to call before the RangeDB is useable.

complete

`complete($database))` returns 1 if we have all relevant data we need to fully reconstruct the F(...) function.

constant

`constant($database)` returns undef if we are not complete, or if different values have been inserted. If all data is complete and the same, returns that value.

translate_from_cube_coords

`translate_from_cube_coords($database,$cube_coord)` returns a hash reference. The hash reference has keys with the same name as the database axes, and values which will be at one end or the other of an axis range.

missing

`missing($database)` returns a hash which represents something in the domain of F which we haven't yet stored.

set

`set($database,$where,$value)` records that F($where) has the value $value. This might cause all sorts of things to happen in the background.

get

`get($database,$where)` returns the value associated with $where by some previous `set($database,$where,$value)` operation.

mitosis

`mitosis($database,$axis_name,$axis_mark)` splits the child Hypercube into two RangeDB objects.

dump

The usual

export_ranges_into

`export_info($database,$exportlist)` appends on to the list reference `$exportlist` a list of hashes containing ranges and values.

export_csv

`export_csv($database,$fh,[$separator]` writes out `$database` into CSV format, (or TSV format if $separator is a tab character).

The axis_names will be used as column headers.

```
"axis_name1 (low)","axis_name2 (high)","axis_name2 (low)",VALUE
10,20,0,10,0.6
10,20,11,50,0.7
21,50,0,50,0.8
```

import_csv

`import_csv($datafile)` reads in a file in the same format as produced by export_csv and returns a RangeDB object.

4.16 ScenarioTags

This package creates a unique sequential identifier number for each scenario.

id_of_scenario

`id_of_scenario($cache,$storage_paths,$scenario)` returns a unique serial number (using the Counter and ScenarioID parameters stored by `$storage_paths`) for `$scenario`.

fudge_scenario_id

`fudge_scenario_id($cache,$storage_paths,$scenario)` updates `$scenario` so that there is also a tag saying "ScenarioID".

4.17 StoragePaths

There are several different paths defined in the [storage] section of a *jones* configuration file.

- **Path**

  Where fetched results will get stored. This will be substituted with any discrete **scenario-based** substitution variables (not ranges), and any **time-based** substitution variables (but not historical ones).

  In fact, for *jones* to work properly, *all* discrete scenario based variables and at least some time-based substitution variables *must* appear in the path somewhere.

- **Log**

  The log of each time *jones* has been run and performed a fetch based on this configuration file. No variables are substituted here at all.

- **ScenarioId**

  Each scenario has a unique identifier number associated with it. They can be different between configuration files, or common across them.

  All **scenario-based** substitution variables (including ranges) which are not fixed must appear. Any others can appear.

  i.e. If you fetch data for two zones, then %z must appear; if only one, then %z is optional.

- **Counter**

  A file to store the largest used scenario id number. No substitutions are done on this.

new

`new($class,$cfg)` takes a Config::IniFiles object and reads the section marked [storage].

timelog_file

`timelog_file($sp)` returns the Log file path, or undef if it wasn't defined.

data_storage

`data_storage($sp,$scenario[,@historical_times])` returns the [storage] Path field with all substitutions already performed.

id_storage

`id_storage($sp,$scenario)` returns the [storage] ScenarioId field with all substitutions already performed.

counter_filename

`counter_filename($sp)` returns the [storage] Counter field, or undef if it wasn't defined.

Other functions

The following functions aren't method functions for StoragePath objects.

translate_path

`translate_and_prepare_path($path_template)` returns a filename with all time-based substitutions performed.

prepare_path

`prepare_path($filename)` makes any needed parent directories, so that open(FILE,">$filename") has a chance of succeeding.

4.18 RangeDB

This package implements an odd kind of database.

Suppose you have a function of many variables F(a,b,c,...,z), and you also know that for any variable V, dF/dV = 0 only once for fixed values of the other variables, or otherwise dF/dV = 0 for all V.

That is, if you took a slice of F with nearly every variable fixed, and graphed the differing values F took as you varied just one variable, the picture would be one of:

a monotonic function

a parabola

a half sinusoid

a hill

a valley

Now suppose it was quite expensive to compute F(a,b,c,...z). What you can do is take a few sample points and see if there are any equi-valued planes or lines through the data. If there are, then you don't need to calculate every value along such a line – you can assume that they are equal just based on the two points you've got so far. Later you can come back and check to see if you might have stripped the top off a peak, or filled the bottom of a valley.

This package implements the kind of database structure you need to hold this kind of data.

new

`new()` creates a new database instance. It takes as an argument the name of each axis.

e.g.

```
new RangeDB('age','weight','height');
```

splinter

`splinter RangeDB ($parent,$split_axis,$replaced_lower,$replaced_higher,$hypercube)` is a constructor

set_axis_range

`set_axis_range($database,$axis_name,$low_value,$high_value)` is one of the essential functions to call before the RangeDB is useable.

complete

`complete($database))` returns 1 if we have all relevant data we need to fully reconstruct the F(...) function.

constant

`constant($database)` returns undef if we are not complete, or if different values have been inserted. If all data is complete and the same, returns that value.

translate_from_cube_coords

`translate_from_cube_coords($database,$cube_coord)` returns a hash reference. The hash reference has keys with the same name as the database axes, and values which will be at one end or the other of an axis range.

missing

`missing($database)` returns a hash which represents something in the domain of F which we haven't yet stored.

set

`set($database,$where,$value)` records that F($where) has the value $value. This might cause all sorts of things to happen in the background.

get

`get($database,$where)` returns the value associated with $where by some previous `set($database,$where,$value)` operation.

mitosis

`mitosis($database,$axis_name,$axis_mark)` splits the child Hypercube into two RangeDB objects.

dump

The usual

export_ranges_into

`export_info($database,$exportlist)` appends on to the list reference `$exportlist` a list of hashes containing ranges and values.

export_csv

`export_csv($database,$fh,[$separator]` writes out `$database` into CSV format, (or TSV format if $separator is a tab character).

The axis_names will be used as column headers.

```
"axis_name1 (low)","axis_name2 (high)","axis_name2 (low)",VALUE
10,20,0,10,0.6
10,20,11,50,0.7
21,50,0,50,0.8
```

import_csv

`import_csv($datafile)` reads in a file in the same format as produced by export_csv and returns a RangeDB object.

4.19 Validator

This module is not documented yet.

4.20 VehicleTags

There are two lines in a jones configuration file section which are a little complicated to work with. They are:

Vehicle

VehicleAge / VehicleYear

Unlike most other jones configuration file options – each of which just turns into a simple tag – these configuration options turn into multiple tags.

And if we are doing range-based searches, then they turn into multiple tags which we need to be able to reconstruct later.

age_field

`age_field($cfg)` returns the VehicleAge or VehicleYear tag, whichever is defined. `$cfg` needs to be a Config::IniFiles object.

tags_list

`tags_list($cfg[,$expand_ranges])` returns a list reference which should be inserted into combinatorix input representing the discrete alternatives for vehicles.

If the optional argument `$expand_ranges` is something true, then ranges will get turned into discrete sets.

insert_range_fields

`insert_range_fields($cfg,$range_fields)` takes the Vehicle and VehicleAge fields from $cfg and puts the minimum and maximum values into the hash reference $range_fields.

If VehicleAge is discrete (and not a range), then this function will do nothing.

fudge_vehicle_variables

`fudge_vehicle_variables($hashref)` looks in the hash ref to figure out if it can figure out some variables (%k, %m and %a) for a vehicle.

4.21 YearComprehension

Ages are specified in four different ways in jones configuration files.

- 1996-2006
- 0-10
- 1996,1997,1998,2001,2005,2009
- 0,1,2,3,5,10

Four digit numbers are obviously the year anno domino and is often used for date of manufacture, or driver's license year.

Any other numbers represent number of years in the past from now.

Internally to jones we use the number of years in the past. When querying the MAACTP website, we turn that back into year of manufacture.

This is simpler because it means we don't need to change the configuration file every year.

years_ago

`years_ago($config_str)` turns a `$config_str` which uses CCYY format into "years ago" format.

I wish I had had this before I wrote `age_min_max`

age_min_max

`age_min_max($config_val)` parses a jones configuration line item and returns the minimum and maximum years old.

is_range

`is_range($config_val)` parses a jones configuration file line and says whether or not the age specified is a range.

Chapter 5

Sample Configs

5.1 full.jns

```
# Configuration for a large, full run to store lots of premiums
[storage]
Path=.jones/jones-full-%D-%B-%X%Y/%H:%T/%k-%z-%h-%o-%u-%e-%c-%r-%I.db
Log=.jones/jones-full.log

[scenarios]
VehicleAge=0-20
Vehicle=Toyota Prius,Ford Falcon XR8 Pursuit,250cc Kawasaki Ninja
Zone=Metro,Country,Newcastle,Wollongong,Outer
DriverAge=18-80
History=Good Driver,Bad Driver
Ownership=Privately Owned,Company Owned
Usage=Private Usage,Business Usage
Existing=CTP with Allianz,No CTP
Comprehensive=No comprehensive
Gender=Male
Commencement=0
Roadside=NRMA Member,No roadside assistance

# No export or changes section. I'm expecting that this would
# be used in conjunction with a second configuration file which
# have
#
# [fetch]
# enabled=no
#
# in it somewhere.

# Without an export section, the historical section is irrelevant.
[historical]
0=previous
```

```
1=1 month ago
2=2 months ago
3=3 months ago
4=6 months ago
5=1 year ago
```

5.2 alerts.jns

```
# Configuration file for a quick run to identify pricing movements.
[storage]
Path=.jones/alerts/%H:%T/%z/%h/%c/%I.price
Log=.jones/alerts.log

[fetch]
threads=1

[scenarios]
VehicleAge=0
Vehicle=Toyota Prius (class 1)
Zone=Metro,Country
DriverAge=50
History=Good Driver and NRMA Member,Bad Driver without Roadside Assistance
Ownership=Privately Owned
Usage=Private Usage
Existing=Lapsed CTP
Comprehensive=Comprehensive with Allianz,No comprehensive
Gender=Female
Commencement=60

[changes]
Format=%I changed from %0P to %P for a %v in the %z zone for a %h with %c (commencing %R/%U/%V%W).
Header=PREMIUM PRICING MOVEMENTS ALERT between %0D-%0B-%0X%0Y %0H:%0T and %D-%B-%X%Y %H:%T
Outfile=.jones/alerts/%D-%B-%X%Y/%H:%T.txt
Command=cp .jones/alerts/%D-%B-%X%Y/%H:%T.txt /var/www/vhosts/www.ifost.org.au/jones/data/latest-change.txt
SMSusername=XXXXYXYX
SMSpassword=zyzyZyZYX
SMSphones=0408123456
SMShead=Premium changes. %R/%U/%V%W. Insurers:
```

```
SMStexts= %I
SMTPServer=mail.ifost.org.au
FromAddress=jones@ifost.org.au
ToAddress=gregb@ifost.org.au
EmailSubject=Premium changes between %0D-%0B-%0X%0Y %0H:%0T and %D-%B-%X%Y %H:%T
EmailLines=%I changed from %0P to %P for a %v in the %z zone for a %h with %c (commencing %R/%U/%V%W).
```

Appendix A

Procedure for adding another field

The tags "Good Driver" and "Bad Driver" supply answers to many questions. If you want to have a more fine-grained approach, it involves modifying the program source.

1. In `perllib/JonesConfig.pm`, in `read_scenarios_section`, add the extra field to `@discrete_fields`.
2. Still in `perllib/JonesConfig.pm` find `%Substitutions` and find a percentage-string which hasn't already been used. Add that extra letter as a key, with the value being the string used in `@discrete_fields`
3. In `perllib/MAACTP.pm` find the repeated calls to `make_tag`. Remove any question-answer pairs which should no longer be in a big tag (e.g. "Good Driver" or "Bad Driver"), and create a new `make_tag` call with the new field (and whatever question-answer pairs it supplies).
4. Edit `doc/jns.pm` in the `[scenarios]` section. Add another `=item` with the name used in `@discrete_fields` stating that the possible values are the tag names from `perllib/MAACTP`.
5. Still in `doc/jns.pm`, in the Scenario-based substitution section, add another `=item` with the percentage-string (from `JonesConfig::Substitutions`) and the name from `@discrete_fields`.
6. Now edit all the files under `conf/` which will be affected by this change (which will probably be all of them), and add a line to the `[scenarios]` section of each of them with the new field. If a config file now uses several alternative values for this new field, then you are likely to need to alter the `Path` parameter in `[storage]`.

Appendix B

Working with Microsoft's ISA proxy and firewall

Some organisations have chosen to use Microsoft's proprietary proxy and firewall application, called "Microsoft ISA" – also known as Microsoft's Internet Security and Accelerator Server.

This software requires users to connect with Microsoft's own proprietary protocol to the proxy server. Usually, authentication is done by username and password against an ActiveDirectory database, which often means the password will need to change every three months or so.

For this reason, when `jones` is run inside an organisation using Microsoft ISA there is an extra step to perform which is done by default as part of the installer – which is to create `C:\jones\proxy.cfg` and edit the username, password, domain and proxy fields.

When you change your password, you will need to change your password in this file as well.

This functionality is only available on MS-Windows. For users running `jones` on Linux or Unix behind an ISA proxy server, install and run `cntlm` from `http://cntlm.sourceforge.net/`.

If you do not wish to store plain text passwords in the configuration file – and there is no alternative proxy server available – there is some documentation on storing hashes in the `cntlm` documentation at the same website.

Index