class Config::DataLang::Refine

Refine use of DataLang configuration


```
class Config::DataLang::Refine { ... }
```

Synopsis

The following piece of code

```
use Config::DataLang::Refine;

my Config::DataLang::Refine $c .= new(:config-name<myConfig.toml>);

my Hash $hp1 = $c.refine(<options plugin1 test>);
my Hash $hp2 = $c.refine( <options plugin1 test>, :filter);
my Array $ap3 = $c.refine-str( <options plugin1 deploy>, :filter);
my Array $ap4 = $c.refine-str( <options plugin2 deploy>, :filter);
```

With the following config file in myConfig.toml

```
[options]
  key1 = 'val1'
  key1a = true

[options.plugin1]
  key2 = 'val2'

[options.plugin1.test]
  key1 = false
  key2 = 'val3'

[options.plugin2.deploy]
  key3 = 'val3'
  key4 = [ 1, 2, 3, 4]
```

Will get you the following as if the variables were set like

```
# All found values
$hp1 = ${:!key1, :key1a, :key2("val3")};

# False booleans filtered out
$hp2 = ${:key1a, :key2("val3")};

# Note that there is no deploy for plugin1
$ap3 = $["key1=val1", "key1a", "key2=val2"];

# Arrays become comma separated lists by default
$ap4 = $["key1=val1", "key1a", "key3=val3", "key4=1,2,3,4"]
```

Description

This class is used for getting configuration data in such a way that several levels are accumulated into a single level Hash or Array. The top level of the configuration should always be a Hash (at this moment).

Attributes

config

Defined as

```
has Hash $.config;
```

Stored configuration. Can be retrieved directly from object.

```
my $c = Config::DataLang::Refine.new;
$c.config<some-key><other-key>;
```

Methods

new

Defined as

```
submethod BUILD (
  Str :$config-name,
  Bool :$merge = False,
  Array :$locations = [],
  Str :$data-module = 'Config::TOML'
)
```

Reads configuration text from a file pointed to by **:config-name**. The file will first be searched for in the current directory. Then, if not found, tries to read the hidden variant (on unixes) which is the name with a dot ('.') prefixed to the file. If that fails too it tries yet another file (also hidden) located in the home directory of the user. At last the method throws an exception if no files are found. If **:config-name** is not defined the program name is taken where the extension is substituted by the proper name for the configuration language.

When **:locations** is defined the array will be used as extra paths to search for the config file. Example paths to add are /etc on unixes or C:/Program Files/MyApp on windows.

When **:config-name** is a relative or absolute path to a config file, then the basename is taken and the path to the file is pushed on the **:locations** array.

:merge is used to merge all the files together starting with the file in the users first and following paths from **:locations**, Then the one from the home directory if found. Then the options from the hidden local file if found and finishing with the visible local file found. An exception will be thrown when the resulting config has no elements.

The data languages such as Config::TOML might throw exceptions when it fails to parse the configuration text.

```
Setup                          Search
============================== ============================================
Nothing set                    :config-name set to program name. Say p.pl6 so
                               config will be p.toml because :data-module is
                               by default C<Config::TOML>.
                               Search;
                               p.toml, .p.toml, <home-dir>/.p.toml

:data-module = JSON::Fast      Same as above except extension is .json.
                               Search;
                               p.json, .p.json, <home-dir>/.p.json

:config-name = x.cfg           Search;
                               x.cfg, .x.cfg, <home-dir>/.x.cfg

:config-name = ../pqr/x.cfg    While shown in the search still as a relative
                               path, the path will be made absolute.
                               Search;
                               x.cfg, .x.cfg, <home-dir>/.x.cfg, ../pqr/x.cfg

:config-name = x.cfg           Search;
:locations = [/etc, /opt/etc] x.cfg, .x.cfg, <home-dir>/.x.cfg, /etc/x.cfg
                               /opt/etc/x.cfg
```

When :merge is used the search is started at the end of the list ending at the first file.

refine

Defined as

```
method refine ( *@key-list, Bool :$filter = False --> Hash )
```

Processes data in the config using the keys from the @key-list. The method returns a single level Hash.

The process starts with taking the first key from the list and gathers all pairs ignoring pairs of which the value is a Hash. Then it descends in the config using the second key. This goes on until the last key is used. The process stops when a key does not exist on some level.

A simple filter is used on the results if **:filter** is set. All key/value pairs are removed from the result where the value is a Bool and is False.

```
type          :!filter        :filter
============= =============== ================
Bool          :k              :k
              :!k             <removed>
Any           :k => v         :k => v
```

refine-str

Defined as

```
method refine-str (
  *@key-list,
  Str :$glue = ',',
  Bool :$filter = False
  StrMode :$str-mode = C-URI-OPTS-T1
  --> Array
)
```

Each string is pushed on the array which is returned.. The **:glue** is the string used to join elements of an array,

this is a ',' by default.

```
:str-mode = C-URI-OPTS-T1 which is the default
type           :!filter         :filter
============== ================ ================
Bool           k=True           k=True
               k=False          <removed>
Array          k=1,2,3          k=1,2,3
special text   k='v'            k='v'
Any            k=v              k=v
```

Simple results

```
:str-mode = C-URI-OPTS-T2
type           :!filter         :filter
============== ================ ================
Bool           k=True           k=True
               k=False          <removed>
Array          k=1,2,3          k=1,2,3
Any            k=v              k=v
```

The results from C-URI-OPTS-T2 can be used to form uri strings when joined together with a '&' character. All strings will be encoded for the first 128 characters of the ascii table.

```
:str-mode = C-UNIX-OPTS-T1
type           :!filter         :filter
============== ================ ================
Bool           --k              --k
               --nok            <removed>
Array          --k=1,2,3        --k=1,2,3
spaced text    --k='v'          --k='v'
Any            --k=v            --k=v
```

All single letter keys get only one dash on front like -k, -nok or -k=v. The mode C-UNIX-OPTS-T2 does the same but gathers all single character keys without values are gatheres together. E.g. --key, -l, -m, -t=1 becomes --key, -lm, -t=1