pyvcd Documentation

Peter Grayson and Steven Sprouse

Jun 19, 2023

Contents

Contents:

CHAPTER 1

vcd

Value Change Dump (VCD) file support.

VCDPhaseError	Indicating a VCDWriter method was called in the wrong phase.
VCDWriter(file, timescale, Tuple[int, str], ...)	Value Change Dump writer.
tokenize(stream, io.RawIOBase], buf_size)	Parse VCD stream into tokens.

CHAPTER 2

vcd.reader

Read Value Change Dump (VCD) files.

The primary interface is the *tokenize()* generator function, parses a binary VCD stream, yielding tokens as they are encountered.

```
>>> import io
>>> from vcd.reader import TokenKind, tokenize
>>> vcd = b"$date today $end $timescale 1 ns $end"
>>> tokens = tokenize(io.BytesIO(vcd))
>>> token = next(tokens)
>>> assert token.kind is TokenKind.DATE
>>> assert token.date == 'today'
>>> token = next(tokens)
>>> assert token.kind is TokenKind.TIMESCALE
>>> assert token.timescale.magnitude.value == 1
>>> assert token.timescale.unit.value == 'ns'
```

vcd.reader.**tokenize**(*stream: Union[io.BufferedIOBase, io.RawIOBase], buf_size: Optional[int] = None*) → Iterator[vcd.reader.Token]

Parse VCD stream into tokens.

The input stream must be opened in binary mode. E.g. with `open(path, 'rb')`.

class vcd.reader.**VCDParseError**(*loc: vcd.reader.Location*, *msg: str*)

Catch-all error for any VCD parsing errors.

loc = None

Location within VCD file where error was detected.

class vcd.reader.**Token**

VCD token yielded from *tokenize()*.

These are relatively high-level tokens insofar as each token fully captures an entire VCD declaration, command, or change descriptor.

The *kind* attribute determines the *data* type. Various kind-specific properties provide runtime type-checked access to the kind-specific data.

Note: The *data* attribute may be accessed directly to avoid runtime type checks and thus achieve better runtime performance versus accessing kind-specific properties such as *scalar_change*.

kind
The kind of token.

span
The start and end location of the token within the file/stream.

data
Data associated with the token. The data type depends on *kind*.

comment
Unstructured text from a `$comment` declaration.

date
Unstructured text from a `$date` declaration.

scope
Scope type and identifier from `$scope` declaration.

timescale
Magnitude and unit from `$timescale` declaration.

var
Details from a `$var` declaration.

version
Unstructured text from a `$version` declaration.

time_change
Simulation time change.

scalar_change
Scalar value change descriptor.

vector_change
Vector value change descriptor.

real_change
Real (float) value change descriptor.

string_change
String value change descriptor.

class vcd.reader.**TokenKind**
Kinds of VCD tokens.

COMMENT = 1

DATE = 2

ENDDEFINITIONS = 3

SCOPE = 4

TIMESCALE = 5

UPSCOPE = 6

VAR = 7

VERSION = 8

DUMPALL = 9

DUMPOFF = 10

DUMPON = 11

DUMPVARS = 12

END = 13

CHANGE_TIME = 14

CHANGE_SCALAR = 15

CHANGE_VECTOR = 16

CHANGE_REAL = 17

CHANGE_STRING = 18

class vcd.reader.**VarDecl**

VCD variable declaration.

Examples:

```
$var wire 4 !@# foobar [ 3 : 1 ] $end
$var real 1 aaa foobar $end
$var integer 32 > foobar[8] $end
```

type_

Type of variable

size

Size, in bits, of variable

id_code

Identifer code of variable.

This code is used in subsequent value change descriptors to map-back to this variable declaration.

reference

Reference name of variable.

This human-readable name typically corresponds to the name of a variable in the model that output the VCD.

bit_index

Optional range of bits to select from the variable.

May select a single bit index, e.g. `ref [ 3 ]`. Or a range of bits, e.g. from `ref [ 7 :  3 ]` (MSB index then LSB index).

class vcd.reader.**ScopeDecl**

VCD scope declaration.

Examples:

```
$scope module Foo $end
$scope
   fork alpha_beta
$end
```

type_

Type of scope

ident
Scope name

class vcd.reader.**ScalarChange**
Scalar value change descriptor.

A scalar is a single 4-state value. The value is one of '0', '1', 'X', or 'Z'.

id_code
Identifier code of associated variable.

value
New value of associated scalar variable.

class vcd.reader.**VectorChange**
Vector value change descriptor.

A vector value consists of multiple 4-state values, where the four states are 0, 1, X, and Z. When a vector value consists entirely of 0 and 1 states, *value* will be an int. Otherwise *value* will be a str.

id_code
Identifier code of associated variable.

value
New value of associated vector variable.

class vcd.reader.**RealChange**
Real value (floating point) change descriptor.

id_code
Identifier code of associated variable.

value
New value of associated real variable.

class vcd.reader.**StringChange**
String value change descriptor.

Strings are VCD extension supported by GTKWave.

id_code
Identifier code of associated variable.

value
New value of associated string variable.

class vcd.reader.**Location**
Describe location within VCD stream/file.

line
Line number

column
Column number

class vcd.reader.**Span**
Describe location span within VCD stream/file.

start
Start of span

end
End of span

CHAPTER 3

vcd.writer

Write Value Change Dump files.

This module provides *VCDWriter* for writing VCD files.

class vcd.writer.**VCDPhaseError**
Indicating a *VCDWriter* method was called in the wrong phase.

For example, calling `register_var()` after `close()` will raise this exception.

class vcd.writer.**VCDWriter**(*file: IO[str], timescale: Union[vcd.common.Timescale, Tuple[int, str], str] = '1 us', date: Optional[str] = None, comment: str = '', version: str = '', default_scope_type: Union[vcd.common.ScopeType, str] = <ScopeType.module: 'module'>, scope_sep: str = '.', check_values: bool = True, init_timestamp: Union[int, float] = 0*)
Value Change Dump writer.

A VCD file captures time-ordered changes to the value of variables.

Parameters

- **file** (*file*) – A file-like object to write the VCD data.
- **timescale** (*str, tuple*) – Scale of the VCD timestamps. The timescale may either be a string or a tuple containing an (int, str) pair.
- **date** (*str*) – Optional *$date* string used in the VCD header.
- **comment** (*str*) – Optional *$comment* string used in the VCD header.
- **version** (*str*) – Optional *$version* string used in the VCD header.
- **default_scope_type** (*str*) – Scope type for scopes where *set_scope_type()* is not called explicitly.
- **scope_sep** (*str*) – Separator for scopes specified as strings.
- **init_timestamp** (*int*) – The initial timestamp. default=0

Raises **ValueError** – for invalid timescale values

set_scope_type(*scope: Union[str, Sequence[str]], scope_type: Union[vcd.common.ScopeType, str]*) → None
Set the scope_type for a given scope.

The scope's type may be set to one of the valid `ScopeType` values. VCD viewer applications may display different scope types differently.

Parameters

- **scope** (*str or sequence of str*) – The scope to set the type of.
- **scope_type** (*str*) – A valid scope type string.

Raises **ValueError** – for invalid *scope_type*

register_var(*scope: Union[str, Sequence[str]], name: str, var_type: Union[vcd.common.VarType, str], size: Union[int, Sequence[int], None] = None, init: Union[bool, int, float, str, None, Sequence[Union[int, bool, str, None]]] = None*) → vcd.writer.Variable
Register a new VCD variable.

All VCD variables must be registered prior to any value changes.

Parameters

- **scope** (*str or sequence of str*) – The hierarchical scope that the variable belongs within.
- **name** (*str*) – Name of the variable.
- **var_type** (`VarType`) – Type of the variable.
- **size** (*int or tuple(int) or None*) – Size, in bits, of the variable. The *size* may be expressed as an int or, for vector variable types, a tuple of int. When the size is expressed as a tuple, the *value* passed to *change()* must also be a tuple of same arity as the *size* tuple. Some variable types ('integer', 'real', 'realtime', and 'event') have a default size and thus *size* may be `None` for those variable types.
- **init** – Optional initial value; defaults to 'x'.

Raises

- ***VCDPhaseError*** – if any values have been changed
- **ValueError** – for invalid var_type value
- **TypeError** – for invalid parameter types
- **KeyError** – for duplicate var name

Returns *Variable* instance appropriate for use with *change()*.

register_alias(*scope: Union[str, Sequence[str]], name: str, var: vcd.writer.Variable*) → None
Register a variable alias.

The same VCD identifier may be associated with multiple reference names ("$var" declarations). This method associates an existing *Variable* instance with a different variable scope and/or name. The alias shares the same identifier, type, size, and value as the reference variable. Because the identifier is shared, calling *change()* with `var` changes the value of of all associated reference names.

Parameters

- **scope** (*str or sequence of str*) – The hierarchical scope that the variable belongs within.
- **name** (*str*) – Name of the variable.
- **var** (`Variable`) – Existing variable to alias.

dump_off (*timestamp: Union[int, float]*) → None
Suspend dumping to VCD file.

dump_on (*timestamp: Union[int, float]*) → None
Resume dumping to VCD file.

change (*var: vcd.writer.Variable, timestamp: Union[int, float], value: Union[bool, int, float, str, None, Sequence[Union[int, bool, str, None]]]*) → None
Change variable's value in VCD stream.

This is the fundamental behavior of a *VCDWriter* instance. Each time a variable's value changes, this method should be called.

The *timestamp* must be in-order relative to timestamps from previous calls to *change()*. It is okay to call *change()* multiple times with the same *timestamp*, but never with a past *timestamp*.

Note: *change()* may be called multiple times before the timestamp progresses past 0. The last value change for each variable will go into the $dumpvars section.

Parameters

- **var** (*Variable*) – *Variable* instance (i.e. from *register_var()*).
- **timestamp** (*int*) – Current simulation time.
- **value** – New value for *var*. For *VectorVariable*, if the variable's *size* is a tuple, then *value* must be a tuple of the same arity.

Raises

- **ValueError** – if the value is not valid for *var*.
- ***VCDPhaseError*** – if the timestamp is out of order or the *VCDWriter* instance is closed.

close (*timestamp: Union[int, float, None] = None*) → None
Close VCD writer.

Any buffered VCD data is flushed to the output file. After *close()*, no variable registration or value changes will be accepted.

Parameters **timestamp** (*int*) – optional final timestamp to insert into VCD stream.

Note: The output file is not automatically closed. It is up to the user to ensure the output file is closed after the *VCDWriter* instance is closed.

---

flush (*timestamp: Union[int, float, None] = None*) → None
Flush any buffered VCD data to output file.

If the VCD header has not already been written, calling *flush()* will force the header to be written thus disallowing any further variable registration.

Parameters **timestamp** (*int*) – optional timestamp to insert into VCD stream.

class vcd.writer.**Variable** (*ident: str, type: vcd.common.VarType, size: Union[int, Sequence[int]], init: ValueType*)
VCD variable details needed to call *VCDWriter.change()*.

ident
Identifier used in VCD output stream.

type
VCD variable type; one of `VCDWriter.VAR_TYPES`.

size
Size, in bits, of variable.

value
Last value of variable.

format_value(*value: ValueType*, *check: bool = True*) → str
Format value change for use in VCD stream.

class vcd.writer.**ScalarVariable**(*ident: str, type: vcd.common.VarType, size: Union[int, Sequence[int]], init: ValueType*)
One-bit VCD scalar.

This is a 4-state variable and thus may have values of 0, 1, 'z', or 'x'.

format_value(*value: Union[int, bool, str, None]*, *check: bool = True*) → str
Format scalar value change for VCD stream.

Parameters **value** (*str, bool, int, or None*) – 1-bit (4-state) scalar value.

Raises **ValueError** – for invalid *value*.

Returns string representing value change for use in a VCD stream.

class vcd.writer.**RealVariable**(*ident: str, type: vcd.common.VarType, size: Union[int, Sequence[int]], init: ValueType*)
Real (IEEE-754 double-precision floating point) variable.

Values must be numeric and cannot be 'x' or 'z' states.

format_value(*value: Union[float, int]*, *check: bool = True*) → str
Format real value change for VCD stream.

Parameters

- **value** – Numeric changed value.
- **type** – float or int

Raises **ValueError** – for invalid real *value*.

Returns string representing value change for use in a VCD stream.

class vcd.writer.**VectorVariable**(*ident: str, type: vcd.common.VarType, size: Union[int, Sequence[int]], init: ValueType*)
Bit vector variable type.

This is for the various non-scalar and non-real variable types including integer, register, wire, etc.

format_value(*value: Union[int, bool, str, None]*, *check: bool = True*) → str
Format value change for VCD stream.

Parameters **value** – New value for the variable.

Types value int, str, or None

Raises **ValueError** – for *some* invalid values.

A *value* of *None* is the same as *'z'*.

> **Warning:** If *value* is of type `str`, all characters must be one of *'01xzXZ'*. For the sake of performance, checking **is not** done to ensure value strings only contain conforming characters. Thus it is possible to produce invalid VCD streams with invalid string values.

class vcd.writer.**CompoundVectorVariable**(*ident: str, type: vcd.common.VarType, size: Union[int, Sequence[int]], init: ValueType*)

Bit vector variable type with a compound size.

This is for the various non-scalar and non-real variable types including integer, register, wire, etc.

format_value(*value: Sequence[Union[int, bool, str, None]], check: bool = True*) → str

Format value change for VCD stream.

Parameters **value** – Sequence of scalar components of the variable's value. The sequence must be the same length as the variable's size tuple.

Returns string representing value change for use in a VCD stream.

CHAPTER 4

vcd.common

class vcd.common.**ScopeType**
Valid VCD scope types.

begin = 'begin'

fork = 'fork'

function = 'function'

module = 'module'

task = 'task'

class vcd.common.**VarType**
Valid VCD variable types.

event = 'event'

integer = 'integer'

parameter = 'parameter'

real = 'real'

realtime = 'realtime'

reg = 'reg'

supply0 = 'supply0'

supply1 = 'supply1'

time = 'time'

tri = 'tri'

triand = 'triand'

trior = 'trior'

trireg = 'trireg'

tri0 = 'tri0'

tri1 = 'tri1'

wand = 'wand'

wire = 'wire'

wor = 'wor'

string = 'string'

class vcd.common.**TimescaleMagnitude**
Valid timescale magnitudes.

one = 1

ten = 10

hundred = 100

class vcd.common.**TimescaleUnit**
Valid timescale units.

second = 's'

millisecond = 'ms'

microsecond = 'us'

nanosecond = 'ns'

picosecond = 'ps'

femtosecond = 'fs'

class vcd.common.**Timescale**
Timescale magnitude and unit.

magnitude
Alias for field number 0

unit
Alias for field number 1

classmethod from_str(*s: str*) → vcd.common.Timescale

CHAPTER 5

vcd.gtkw

GTKWave save file generator.

This module provides tools for generating GTKWave save files.

GTKWave is an application for viewing VCD data. When opening a VCD file with GTKWave, by default, no VCD variables (signals) are displayed. It is thus useful to have an accompanying "save" file which configures various aspects of how GTKWave shows the VCD data, including which variables are displayed, variable aliases, color information, and more.

class vcd.gtkw.**GTKWSave**(*savefile: IO[str]*)

Write GTKWave save files.

This class provides methods for writing the various pieces of a GTKWave save file. A GTKWave save file compliments a VCD dump file with dump file specific configuration GTKWave uses to display the dump file.

A GTKWave save file is line-oriented ASCII text. Each line consists of a single configuration directive. All directives are optional.

Some directives, such as *dumpfile()*, are for general GTKWave configuration. These general directives may be added anywhere in the save file and in any order relative to other directives. Directives may also be duplicated–the last one added will be used by GTKWave.

The *trace()*, *trace_bits()*, *group()*, and *blank()* directives add signals to the "Signals" list which are traced in the "Waves" frame. The order in which these signal traces are added determines the order in GTKWave.

comment(**comments*) → None

Add comment line(s) to save file.

dumpfile(*dump_path: str*, *abspath: bool = True*) → None

Add VCD dump file path to save file.

The *[dumpfile]* must be in the save file in order to only have to specify the save file on the *gtkwave* command line. I.e.:

```
$ gtkwave my.gtkw
```

If the *[dumpfile]* is not present in the save file, both the dump and save files must be specified to *gtkwave*:

$ gtkwave my.vcd my.gtkw

Parameters

- **dump_path** – path to VCD dump file or None to produce special "(null)" value in the save file.
- **abspath** (*bool*) – convert *dump_path* to an absolute path.

dumpfile_mtime(*mtime: Union[float, time.struct_time, datetime.datetime, None] = None*, *dump_path: Optional[str] = None*) → None
Add dump file modification time to save file.

Configuring the dump file's modification time is optional.

dumpfile_size(*size: Optional[int] = None*, *dump_path: Optional[str] = None*) → None
Add dump file size annotation to save file.

Configuring the dump file's size is optional.

savefile(*save_path: Optional[str] = None*, *abspath: bool = True*) → None
Add the path of the save file to the save file.

With no parameters, the output file's name will be used.

Configuring the *[savefile]* is optional.

Parameters

- **save_path** – path to this save file. None will use the output file's path.
- **abspath** (*bool*) – determines whether to make the path absolute.

timestart(*timestamp: int = 0*) → None
Add simulation start time to the save file.

zoom_markers(*zoom: float = 0.0*, *marker: int = -1*, ***kwargs*) → None
Set zoom, primary marker, and markers 'a' - 'z'.

size(*width: int*, *height: int*) → None
Set GTKWave window size.

pos(*x: int = -1*, *y: int = -1*) → None
Set GTKWave window position.

treeopen(*tree: str*) → None
Start with *tree* open in Signal Search Tree (SST).

GTKWave specifies tree paths with a trailing '.'. The trailing '.' will automatically be added if it is omitted in the *tree* parameter.

Parameters **tree** (*str*) – scope/path/tree to be opened in GTKWave's SST frame.

signals_width(*width: int*) → None
Set width of Signals frame.

sst_expanded(*is_expanded: bool*) → None
Set whether Signal Search Tree (SST) frame is expanded.

pattern_trace(*is_enabled: bool*) → None
Enable/disable pattern trace.

group(*name: str*, *closed: bool = False*, *highlight: bool = False*) → Generator[None, None, None]
Contextmanager helper for *begin_group()* and *end_group()*.

This context manager starts a new group of signal traces and ends the group when leaving the *with* block. E.g.:

```
>>> import io
>>> gtkw = GTKWSave(io.StringIO())
>>> with gtkw.group('mygroup'):
...     gtkw.trace('a.b.x')
...     gtkw.trace('a.b.y')
...     gtkw.trace('a.b.z')
```

Parameters

- **name** (*str*) – the name/label of the trace group.
- **closed** (*bool*) – group should be closed at GTKWave startup.
- **highlight** (*bool*) – group should be highlighted at GTKWave startup.

begin_group(*name: str*, *closed: bool = False*, *highlight: bool = False*) → None

Begin a new signal trace group.

Consider using *group()* instead of *begin_group()* and *end_group()*.

Parameters

- **name** (*str*) – the name/label of the trace group.
- **closed** (*bool*) – group should be closed at GTKWave startup.
- **highlight** (*bool*) – group should be highlighted at GTKWave startup.

end_group(*name: str*, *closed: bool = False*, *highlight: bool = False*) → None

End a signal trace group.

This call must match with a prior call to `begin_group()`. Consider using :meth:`group()` instead of *begin_group()* and *end_group()*.

Parameters

- **name** (*str*) – the name/label of the trace group.
- **closed** (*bool*) – group should be closed at GTKWave startup.
- **highlight** (*bool*) – group should be highlighted at GTKWave startup.

blank(*label: str = ''*, *analog_extend: bool = False*, *highlight: bool = False*) → None

Add blank or label to trace signals list.

Parameters

- **label** (*str*) – Optional label for the blank.
- **analog_extend** (*bool*) – extend the height of an immediately preceding analog trace signal.
- **highlight** (*bool*) – blank should be highlighted at GTKWave startup.

trace(*name: str, alias: Optional[str] = None, color: Union[vcd.gtkw.GTKWColor, str, int, None] = None, datafmt: str = 'hex', highlight: bool = False, rjustify: bool = True, extraflags: Union[vcd.gtkw.GTKWFlag, Sequence[str], None] = <GTKWFlag.0: 0>, translate_filter_file: Optional[str] = None, translate_filter_proc: Optional[str] = None*) → None

Add signal trace to save file.

Parameters

- **name** (*str*) – fully-qualified name of signal to trace.
- **alias** (*str*) – optional alias to display instead of the *name*.
- **color** (GTKWColor) – optional color to use for the signal's trace.
- **datafmt** (*str*) – the format used for data display. Must be one of 'hex', 'dec', 'bin', 'oct', 'ascii', 'real', or 'signed'.
- **highlight** (*bool*) – trace should be highlighted at GTKWave startup.
- **rjustify** (*bool*) – trace name/alias should be right-justified.
- **extraflags** (GTKWFlag) – extra flags to apply to the trace.
- **translate_filter_file** (*str*) – path to translate filter file.
- **translate_filter_proc** (*str*) – path to translate filter executable.

Note: GTKWave versions <= 3.3.64 require vector signal names to have a bit range suffix. For example, an 8-bit vector variable "module.myint" would be known by GTKWave as "module.myint[7:0]".

GTKWave versions after 3.3.64 do not use bit-range suffixes.

---

trace_bits(*name: str, alias: Optional[str] = None, color: Union[str, int, None] = None, datafmt: str = 'hex', highlight: bool = False, rjustify: bool = True, extraflags: Union[vcd.gtkw.GTKWFlag, Sequence[str], None] = <GTKWFlag.0: 0>, translate_filter_file: Optional[str] = None, translate_filter_proc: Optional[str] = None*) → Generator[None, None, None]

Contextmanager for tracing bits of a vector signal.

This allows each individual bit of a vector signal to have its own trace (and trace configuration).

```
>>> import io
>>> gtkw = GTKWSave(io.StringIO())
>>> name = 'mod.myint'
>>> with gtkw.trace_bits(name):
...     gtkw.trace_bit(0, name)
...     gtkw.trace_bit(1, name)
...     gtkw.trace_bit(2, name)
...     gtkw.trace_bit(3, name, 'special', color=GTKWColor.yellow)
```

Parameters

- **name** (*str*) – fully-qualified name of the vector variable to trace.
- **alias** (*str*) – optional alias to display instead of *name*.
- **color** (*int*) – optional trace color.
- **datafmt** (*str*) – format for data display.
- **highlight** (*bool*) – trace should be highlighted at GTKWave startup.
- **rjustify** (*bool*) – trace name/alias should be right-justified.
- **extraflags** (GTKWFlag) – extra flags to apply to the trace.
- **translate_filter_file** (*str*) – path to translate filter file.
- **translate_filter_proc** (*str*) – path to translate filter executable.

trace_bit(*index: int*, *name: str*, *alias: Optional[str] = None*, *color: Union[vcd.gtkw.GTKWColor, str, int, None] = None*) → None

Trace individual bit of vector signal.

This is meant for use in conjunction with *trace_bits()*.

Parameters

- **index** (*int*) – index of bit
- **name** (*str*) – name of parent vector signal.
- **alias** (*str*) – optional alias to display for bit.
- **color** (*int*) – optional color for bit's trace.

class vcd.gtkw.**GTKWFlag**

These are the valid GTKWave trace flags.

highlight = 1
Highlight the trace item

hex = 2
Hexadecimal data value representation

dec = 4
Decimal data value representation

bin = 8
Binary data value representation

oct = 16
Octal data value representation

rjustify = 32
Right-justify signal name/alias

invert = 64

reverse = 128

exclude = 256

blank = 512
Used for blank, label, and/or analog height

signed = 1024
Signed (2's compliment) data representation

ascii = 2048
ASCII character representation

collapsed = 4096
Used for closed groups

ftranslated = 8192
Trace translated with filter file

ptranslated = 16384
Trace translated with filter process

analog_step = 32768
Show trace as discrete analog steps

analog_interpolated = 65536
Show trace as analog with interpolation

analog_blank_stretch = 131072
Used to extend height of analog data

real = 262144
Real (floating point) data value representation

analog_fullscale = 524288
Analog data scaled using full simulation time

zerofill = 1048576

onefill = 2097152

closed = 4194304

grp_begin = 8388608
Begin a group of signals

grp_end = 16777216
End a group of signals

bingray = 33554432

graybin = 67108864

real2bits = 134217728

ttranslated = 268435456

popcnt = 536870912
Show the population count, i.e. the number of set bits

fpdecshift = 1073741824

class vcd.gtkw.**GTKWColor**
The colors used by GTKWave.

The *cycle* color is special and indicates the GTKWave should cycle through this list of colors, starting from the last selected color.

cycle = -1
Cycle between colors

normal = 0
Default color

red = 1

orange = 2

yellow = 3

green = 4

blue = 5

indigo = 6

violet = 7

vcd.gtkw.**make_translation_filter**(*translations: Sequence[Tuple[Any, ...]]*, *datafmt: str = 'hex'*, *size: Optional[int] = None*) → str
Create translation filter.

The returned translation filter string that can be written to a translation filter file usable by GTKWave.

Parameters

- **translations** – Sequence of 2-tuples *(value, alias)* or 3-tuples *(value, alias, color)*.
- **datafmt** (*str*) – Format to apply to the translation values. This *datafmt* must match the *datafmt* used with *GTKWSave.trace()*, otherwise these translations will not be matched by GTKWave.

Returns Translation filter string suitable for writing to a translation filter file.

vcd.gtkw.**decode_flags**(*flags: Union[str, int]*) → List[str]
Decode hexadecimal flags from GTKWave save file into flag names.

This is useful for understanding what, for example "@802022" means when inspecting a GTKWave save file.

Parameters **flags** – Hexadecimal flags from GTKWave save file; either as an integer or string with hexadecimal characters.

Returns List of flag names

vcd.gtkw.**spawn_gtkwave_interactive**(*dump_path: str*, *save_path: str*, *quiet: bool = False*) → None
Spawn gtkwave process in interactive mode.

A process pipeline is constructed such that the contents of the VCD dump file at *dump_path* are displayed interactively as the dump file is being written (i.e. with *VCDWriter*.

The process pipeline built is approximately equivalent to:

```
$ tail -f dump_path | shmidcat | gtkwave -vI save_path
```

The `tail`, `shmidcat`, and `gtkwave` executables must be found in `$PATH`.

Warning: This function does not work on Windows.

Note: A child python process of the caller will remain running until the GTKWave window is closed. This process ensures that the various other child processes are properly reaped.

Parameters

- **dump_path** (*str*) – path to VCD dump file. The dump file must exist, but be empty.
- **save_path** (*str*) – path to GTKWave save file. The save file will be read immediately by GTKWave and thus must be completely written.
- **quiet** (*bool*) – quiet GTKWave's output by closing its *stdout* and *stderr* file descriptors.

CHAPTER 6

PyVCD

The PyVCD package writes Value Change Dump (VCD) files as specified in IEEE 1364-2005.

Read the documentation.

Visit PyVCD on GitHub.

6.1 Quick Start

```
>>> import sys
>>> from vcd import VCDWriter
>>> with VCDWriter(sys.stdout, timescale='1 ns', date='today') as writer:
...     counter_var = writer.register_var('a.b.c', 'counter', 'integer', size=8)
...     real_var = writer.register_var('a.b.c', 'x', 'real', init=1.23)
...     for timestamp, value in enumerate(range(10, 20, 2)):
...         writer.change(counter_var, timestamp, value)
...     writer.change(real_var, 5, 3.21)
$date today $end
$timescale 1 ns $end
$scope module a $end
$scope module b $end
$scope module c $end
$var integer 8 ! counter $end
$var real 64 " x $end
$upscope $end
$upscope $end
$upscope $end
$enddefinitions $end
#0
```

(continues on next page)

(continued from previous page)

```
$dumpvars
b1010 !
r1.23 "
$end
#1
b1100 !
#2
b1110 !
#3
b10000 !
#4
b10010 !
#5
r3.21 "
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

V

Index

T

U

V

W

Y

Z