Verification Continuum™ Verdi® Python-Based NPI Waveform Writer Model

Version V-2023.12-SP1, March 2024

SYNOPSYS®

Contents

Preface

The Python Based NPI Waveform Writer Model User Guide provides efficient and convenient APIs for external users writing their own waveform file.

Customer Support

For any online access to the self-help resources, you can refer to the documentation and searchable knowledge base available in SolvNetPlus.

To obtain support for your Verdi product, choose one of the following:

- Open a case through SolvNetPlus.

  Go to https://solvnetplus.synopsys.com/s/contactsupport and provide the requested information, including:

  - Product L1 as Verdi
  - Case Type

  Fill in the remaining fields according to your environment and issue.

- Send an e-mail message to verdi_support@synopsys.com.

  Include product name (L1), sub-product name/technology (L2), and product version in your e-mail, so it can be routed correctly.

  Your e-mail will be acknowledged by automatic reply and assigned a Case number along with Case reference ID in the subject (ref:_...:ref).

  For any further communication on this Case via e-mail, send e-mail to verdi_support@synopsys.com and ensure to have the same Case ref ID in the subject header or else it will open duplicate cases.

- You can call for support at:

  https://www.synopsys.com/support/global-support-centers.html

Note:
In general, we need to be able to reproduce the problem in order to fix it, so a simple model demonstrating the error is the most effective way for us to identify the bug. If that is not possible, then provide a detailed explanation of the problem along with complete error and corresponding code, if any/permissible.

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1 Introduction to Python Based NPI

Python-Based NPI APIs support six models. They are as follows:

- Waveform Writer
- Waveform
- Netlist
- Text
- Coverage
- Language

Each model have their own APIs to let you be able to traverse data objects and obtain objects' properties like the existing C-Based or Tcl-Based NPI APIs.

In this guide, the environment setting for using **Python-Based NPI APIs for Waveform Writer** is demonstrated.

Packages and Modules

Packages

The Python-based NPI package name is "pynpi", and it is placed at `$VERDI_HOME/share/NPI/python`.

Modules

There are seven modules inside the "pynpi" package: npisys, lang, netlist, text, cov waveform, and waveformw. The first module, npisys, is the system model for initialization, loading design and exit. The other modules represent language model, netlist model, text model, coverage model, wave model, and waveform writer model respectively

Module Functions and Class Objects

L0 Module Functions

Every module provides some L0 (level 0) functions to let you get the class objects. These functions return a class object or a list of class objects, and they follow the specification of the existing L0 APIs provided in C or Tcl.

L1 Module Functions

Similar to L0 module functions, every module also provides some L1 (level 1) functions to let you get advanced information based on the results obtained by L0 module functions. These functions follows the specification of the existing L1 APIs provided in C or Tcl.

Class Objects

The class object is similar to the so-called handle in NPI C APIs. The most difference is that some basic L0 APIs in C and Tcl will become class method function. These L0 APIs are usually to get integer value, string value, 1-to-1 method to get a handle, and 1-to-many method to get handle iterator.

User Interface and Use Flow

This chapter describes the user interface and use flow for Python-Based NPI APIs for Waveform Writer.

Environment and Library Setting

The python library setting flow of using Python-Based NPI APIs contains four parts:

1. Check your Python's version:

   Python-Based NPI APIs need the Python version greater than 3.6.0.

2. Environment setting for "`VERDI_HOME`" is required for Python-based NPI. Remember to set it well before running program.

3. Add python library path into your python code before loading Python-Based NPI by using the following commands:

   ```
   rel_lib_path = os.environ['VERDI_HOME'] + '/share/NPI/python'

   sys.path.append(os.path.abspath(rel_lib_path))
   ```

4. Import module “npisys” for using the function of NPI initialization and exit from pynpi package.

   ```
   from pynpi import npisys
   ```

5. Import the module you need from pynpi package. For example, if you want to use waveform writer model, you can import the module as follows:

   ```
   from pynpi import waveformw
   ```

6. Note that initialization function `npisys.init()` must be called before writing your code by using any other modules. Also, `npisys.end()` must be called after finishing your code. Following is a simple example to demonstrate how to use waveform writer model by Python-Based NPI APIs.

   Python program to use NPI waveform writer model: (`demo.py`)

   ```
   #!/global/freeware/Linux/2.X/python-3.6.0/bin/python
   import sys, os
   rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
   sys.path.append(os.path.abspath(rel_lib_path))
   from pynpi import npisys
   from pynpi import waveformw
   # Initialize NPI
   if not npisys.init(sys.argv):
   print("Error: Fail to initialize NPI")
   assert 0
   # Load design (if needed, depends on models)
   if not npisys.load_design(sys.argv):
   print("Error: Fail to load design")
   assert 0
   # Beginning of your code here --------------------
   #
   # Example code can be found in later chapters
   #
   # End of your code -------------------------------
   # End NPI
   npisys.end()
   ```

   C shell script to setup environment and execute Python program on 64-bit machine: (`run_demo`)

   ```
   #!/bin/csh -f
   # Setup your $VERDI_HOME here
   setenv VERDI_HOME [YOUR_VERDI_HOME_PATH]
   # run the python program
   # - Input arguments depend on your program design
   # - If loading design is required, you can pass the options like
   ./demo.py -sv demo.v
   ```

To run the files, put the above files in the same directory and execute the `run_demo` C shell script.

```
./run_demo
```

2

Module npisys

This chapter includes the following topics:

- Overview
- L0 APIs

Overview

Module npisys is for setting Python-based NPI. You must call `npisys.init()` before using any other NPI modules and call `npisys.end()` after using any other NPI modules.

L0 APIs

Following are the public L0 APIs for system module:

npisys.init(pyArgvList)

System initialization for Python-Based NPI.

Parameters: **pyArgList (str list)** – input argument list, for example, sys.argv

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example

```
>>>npisys.init(sys.argv)
```

npisys.load_design(pyArgvList)

Load design for Python-Based NPI.

Parameters: *pyArgList (str list)* – input argument list. For example, sys.argv

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example

```
>>>npisys.load_design(sys.argv)
```

---

npisys.end()

Clean NPI-related settings and data.

Parameters: none

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example

```
>>>npisys.end()
```

Feedback

3

Python Based NPI Waveform Writer Model

This chapter includes the following topics:

Abstract

NPI waveform writer model provides efficient and convenient APIs to write your own waveform file.

Quick Start

Environment and library setting:

1. Add python library path using the following commands:

   ```
   rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
   sys.path.append(os.path.abspath(rel_lib_path))
   ```

2. Import `npisys` to use the function of NPI initialization and exit.

   Import `waveformw` to use the APIs of Waveform Model.

   ```
   from pynpi import npisysm
   from pynpi import waveformw
   ```

3. If there exists any error in `LD_LIBRARY_PATH`, add `$VERDI_HOME/share/NPI/lib/linux64` and

   ```
   "$VERDI_HOME/platform/linux64/bin" to LD_LIBRARY_PATH:

   os.environ['LD_LIBRARY_PATH'] =
   os.environ['VERDI_HOME']+'/share/NPI/lib/
   ```

```
linux64:'+os.environ['VERDI_HOME']+'/platform/linux64/
bin:'+os.environ['LD_LIBRARY_PATH']
```

User Interface and Use Flow

The flow of using Waveform writer APIs contains four parts:

- Waveform File
- Create Hierarchy
- Add value to Signals
- Close Waveform File

1. **Create Waveform File**: Obtain a Waveform writer object from API create() .
2. **Create Hierarchy**: describe the scope and signal hierarchy of the Waveform.

   This phase is between the two APIs:

   - FileObj.begin_hierarchy()
   - FileObj.end_heirarchy()
3. **Add value to signals**: After calling FileObj.begin_hierarchy(), you can add value to signals.

   They can also increase time using API FileObj.incr_time().
4. **Close Waveform File**: close the FSDB writer object and free related memory with API close().

Enums

- Enum lists
- Signal's data type Enums

ScalerDtType_e	Waveform writer signal's data type.
GroupType_e	Waveform writer Group data type.

Signal’s Data Type Enums

ScalerDtType_e

class waveformw.**ScalarDtType_e**

Waveform writer signal’s data type.

SvLogic: 1 bit, 4 states (01xz)

SvInteger: 32 bits, 4 states (01xz)

SvReal: 64 bits, real

SvString: string

SvEvent: 1 bit, 2 states (0/1)

VhLogic: 1 bit, 9 states

VhInteger: 32 bits, 4 states (01xz)

VhReal: 64 bits, real

SpLogic: 1 bit, 4 states (01xz)

SpVoltage: 64 bits, real

SpAvgRmsCurrent: 64 bits, real

SpMathematics: 64 bits, real

SvInteger64 : 64 bits, 4 states (01xz)

SvLogic = 0

SvInteger = 1

SvReal = 2

SvString = 3

SvEvent = 4

VhLogic = 5

VhInteger= 6

VhReal = 7

SpLogic = 8

SpVoltage = 9

SpAvgRmsCurrent = 10

SpMathematics = 11

SvInteger64 = 12

GroupType_e

class waveformw.**GroupType_e**

Waveform writer Group data type.

SvStruct : For SystemVerilog Struct.

SvUnion : For SystemVerilog Union.

VhRecord : For VHDL Record.

SvStruct = 0

SvUnion = 1

VhRecord = 2

L0 APIs

File

Function list

waveformw.create(name='novas.fsdb', unit='1ns', beginTime=0)	Create a Waveform file with a given file name in current working directory.
waveformw.close(file)	Close the Waveform file.

Example:

Following is an example showing how to create a waveform file named `novas.fsdb`.

example.py:

```
import sys , os
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import waveform as writer
    # Create a waveform file "novas.fsdb"
    file = writer.create("novas.fsdb", "10ns", 10)
    # close the waveform file
    writer.close(file)
```

Result:

```
You will get a novas.fsdb waveform file in your current directory .
```

waveform writer function

waveformw.create(*name='novas.fsdb', unit='1ns', beginTime*=0)

Create a Waveform file with a given file name in current working directory.

Parameters

- **name** – Name of the waveform file to be opened for writing. If name is omitted, the default value is `novas.fsdb`.
- **unit** – The scale unit of this file. If unit is omitted, the default value is “1ns”. If the scale unit should be set as 10ns,user can give a string “10ns” as a unit for this API. Scale unit should be as [1, 10, 100][s, ms, us, ns, ps, fs].
- **beginTime** – The begin time of simulation for this Waveform file. If beginTime is omitted, the begin time of this file will be set as 0.

Returns:

- File object, is success.
- None, if fail.

Return Type: class waveformw.FileObj(fileObj)

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
```

waveformw.close(*file*)

Close the Waveform file.

Parameters: file – The Waveform file Object FileObj (class waveformw.FileObj(fileObj)) to be closed.

Returns:

- 1, if success.
- 0, if fail.

Return type: int

Examples:

```
>>> print("Close file ret is ", wwaveformw.close(file))
```

Close file ret is 1

class waveformw.FileObj(*fileObj*)

FileObj Function list:

flush()	Flush the data into Waveform file obj without close it.
time()	Get the current time recorded by the specific Waveform file obj.
begin_hierarchy()	Begin hierarchy create phase (do not allow add value to signal in this phase).
end_hierarchy()	Enter value creation phase (do not allow create scopes/signals under value creation phase).
incr_time(time)	Step forward with the volume of time in simulation time.
create_scope(scopeType, name, defName=None)	Create a sub scope under current scope, and then set the new added scope as current scope.
up_scope()	Make current scope go to its parent scope.
dt(dtType, leftRange=None, rightRange=None)	Get predefined scalar data type.
create_group_dt(groupType, nameList, dtList)	Group a list of data types to a composite data type.
create_array_dt(baseDt, leftRange, rightRange)	Use a base data type to create an array data type.
create_enum_dt(label)	Create an enum data type.
leaf_count(baseDt)	Get the leaf count from an Dt object.
scope_name(isFull=True)	Get the scope name/fullname of current scope.
scope_depth()	Get depth of current scope.

create_sig(scalarDtType, name, dir = waveform.DirType_e.DirNone, leftRange = None, rightRange = None)	Create a signal under current scope.
create_sig_by_dt_obj(dtType, name, dir = waveform.DirType_e.DirNone)	Create a signal under current scope.
create_eq_sig(sourceSig, name, dir = waveform.DirType_e.DirNone, dt = None)	Create an equivalent signal. The created signal and source signal will share same value changes.

Example:

writer.py

```
import sys, os
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import waveformw as writer
def test():
    # Create the wavform file named "test.fsdb"
    file = writer.create("test.fsdb", "10ns", 10)
    # Begin the hierarchy create phase
    file.begin_hierarchy()
    # sub-scope create from rootscope
    if
 file.create_scope(writer.waveform.ScopeType_e.SvModule,"SvModule_sig_tes
t" ) is 1:
    print("Scope full name is", file.scope_name())
    # Up scope to the rootscope
    if file.up_scope() is 0:
    print("Up scope fail.")
    # sub-scope create from rootscope
    if
 file.create_scope(writer.waveform.ScopeType_e.SvModule,"Module_test" )
 is 1:
    print("Scope full name is", file.scope_name())
    # sub-scope create from the scope "Module_test"
    if
 file.create_scope(writer.waveform.ScopeType_e.SvModule,"Module_test_2" )
 is 1:
    print("Scope full name is", file.scope_name())
    # create the sig from the scope "Module_test.Module_test_2"
    sig_basic = file.create_sig(writer.ScalarDtType_e.SvLogic,
 "sig_basic", writer.waveform.DirType_e.DirInout)
```

```
    # create an equivalent signal with the source sig
 "Module_test.Module_test_2.sig_basic"
    sig_eq_basic = file.create_eq_sig( sig_basic , "sig_eq_basic")
    # create the sig with range from the scope
 "Module_test.Module_test_2"
    sig_range = file.create_sig(writer.ScalarDtType_e.SvLogic,
 "sig_range", writer.waveform.DirType_e.DirInput, 2, 10)

 file.create_scope(writer.waveform.ScopeType_e.SvModule,"SvModule_name_hi
er" )
    sig_range_2 = file.create_sig(writer.ScalarDtType_e.SvLogic,
 "sig_range_2", writer.waveform.DirType_e.DirInput, 5, 9)
        print("depth is", file.scope_depth())
        print("Scope full name is", file.scope_name())
        print("Scope name is", file.scope_name(False))
    #create the struct sig by dt
    nameList = ["group_name_1","group_name_2"]
    dtList =
 [file.dt(writer.ScalarDtType_e.SvLogic),file.dt(writer.ScalarDtType_e.Sv
Integer) ]
    groupDt = file.create_group_dt(writer.GroupType_e.SvStruct, nameList,
 dtList)
    # End the hierarchy create phase
    file.end_hierarchy()
    # Add value to sig
    print("sigDt[1]'s name", sigDt[1].name())
    sigDt[0].add_value("0", writer.waveform.VctFormat_e.BinStrVal)
    sigDt[1].add_value("10", writer.waveform.VctFormat_e.SintVal)
    if sig_basic.add_value("0", writer.waveform.VctFormat_e.BinStrVal) is
 1:
        print("success to add value")
    sig_range.add_value("101110101",
 writer.waveform.VctFormat_e.BinStrVal)
    sig_range_2.add_value("01001", writer.waveform.VctFormat_e.BinStrVal)
    print("sig_basic name is:", sig_basic.name())
    cur_time = file.time()
        print("Time now:", cur_time)
    ret = file.incr_time(15)
    if ret is 1:
        print("Time now:", file.time())
    file.flush()
    sigDt[0].add_value("X")
    sigDt[1].add_value("20", writer.waveform.VctFormat_e.SintVal)
    sig_basic.add_value("1", writer.waveform.VctFormat_e.BinStrVal)
    ret = file.incr_time(15)
    # Create array dt
    dt1 = file.dt(writer.ScalarDtType_e.SvLogic, 3, 10 )
    ret = file.leaf_count(dt1)
       print("Leaf count", ret)
    arrayDt = file.create_array_dt(dt1, 0, 5)
    ret = file.leaf_count(arrayDt)
       print("Array leaf count", ret)
    # Close waveform file
    if writer.close(file) is 1:
       print("Close waveform file.")
    if __name__ == '__main__':
    orig_stdout = sys.stdout
    f = open('writer.log', 'w')
    sys.stdout = f
    test()
    sys.stdout = orig_stdout
    f.close()
```

Result: writer.log

And a waveform file named “test.fsdb”.

```
Scope full name is SvModule_sig_test
Scope full name is Module_test
Scope full name is Module_test.Module_test_2
depth is 3
Scope full name is Module_test.Module_test_2.SvModule_name_hier
Scope name is SvModule_name_hier
sigDt[1]'s name sigDt.group_name_2
success to add value
sig_basic name is: sig_basic
Time now: 10
Time now: 25
Leaf count 1
Array leaf count 6
Close waveform file.
```

flush()

Flush the data into Waveform file obj without closing it.

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
file.flush()
```

time()

Get the current time recorded by the specific Waveform file obj.

Returns: Return current time.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
print("Time now:", file.time())
Time now: 10
```

begin_hierarchy()

Begin hierarchy create phase (does not allow add value to signal in this phase).

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
file.begin_hierarchy()
# Create the waveform's hierarchical tree.
file.end_hierarchy()
```

end_hierarchy()

Enter value creation phase (do not allow create scopes/signals under value creation phase).

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
file.begin_hierarchy()
# Create the waveform's hierarchical tree.
file.end_hierarchy()
```

incr_time(*time*)

Step forward with the volume of time in simulation time.

Parameters: **time** – Target increase time in the specified file object.

Returns:

- 1, if success.
- 0, if fail.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 10)
ret = file.incr_time(15)
if ret is 1:
print("Time now:", file.time())
Time now: 25
```

create_scope(*scopeType, name, defName=None*)

Create a sub scope under current scope, and then set the new added scope as current scope.

Note:
The following scope types cannot be created:

- ScopeType_e.ScModule
- ScopeType_e.Unknown

(For information on enum `ScopeType_e`, see the *Python-Based NPI Waveform Model* guide.)

Parameters:

- **scopeType** – The scope type. (Refer to the PyNPI Waveform Model's enum `ScopeType_e`)
- **name** – The scope name. (this field cannot be NULL)
- **defName** – The scope define name. (for example, module name) (default is NULL)

Returns:

- 1, if success.
- 0, if fail.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
if
 file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_tes
t" ) is 1:
print("Scope full name is ", file.scope_name())
file.end_hierarchy()
Scope full name is Module_test
```

up_scope()

Make current scope go to its parent scope.

If no parent scope exits, it stays at current scope.

Returns:

- 1, if success.
- 0, if fail or no parent scope

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test_
2" )
if file.up_scope() is 1:
print("Scope full name is ", file.scope_name())
file.end_hierarchy()
Scope full name is Module_test
```

dt(*dtType, leftRange=None, rightRange=None*)

Get predefined scalar data type.

Parameters:

dtType – Basic scalar data type ScalarDtType_e.

leftRange – Left range of vector data type. The default value is None.

rightRange – Right range of vector data type. The default value is None.

Returns:

- A data type object, if success.
- None, if fail.

Return type: dt object

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
dt1 = file.dt(waveformw.waveform.ScopeType_e.SvModule)
print(type(dt1))
<enum 'ScopeType_e'>
```

create_group_dt(*groupType, nameList, dtList*)

Group a list of data types to a composite data type.

Parameters:

- **groupType** – Group type GroupType_e.
- **nameList** – Name list of group items.
- **dtList** – Dt list of group items.

Returns:

- A new data type object, if success.
- None, if fail.

Return type: dt object

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
nameList = ["group_name_1","group_name_2"]
dtList =
 [file.dt(waveformw.ScalarDtType_e.SvLogic),file.dt(waveformw.ScalarDtTyp
e_e.SvInteger) ]
groupDt = file.create_group_dt(ww.GroupType_e.SvStruct, nameList, dtList)
```

create_array_dt(*baseDt, leftRange, rightRange*)

Use a base data type to create an array data type.

Parameters:

baseDt – Base data type.

leftRange – Left-hand-side range.

rightRange – Right-hand-side range.

Returns:

- A new data type object, if success.
- None, if fail.

Return type: dt object

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
arrayDt = file.create_array_dt(dt1, 0, 5)
```

create_enum_dt(label)

Create an enum data type. Users need to provide a Label List or a Lable OrderedDict with enum information.

Parameters:

- **label** – Label List or Label OrderedDict with enum information.
- **List (Label)** – The string type of label elements' name. ["name1", "name2", …….,"nameN"], and the name should be unique. The value of the label increases automatically (start from 0).
- **OrderedDict (Label)** – The OrderedDict keys represent the label elements' name (string type), and the name should be unique. The OrderedDict keys represent the label elements' value, the valid value type are None, "", bitString(01XZ), and positive integer. The value of the label increases automatically (start from 0 or previous countable value), when previous value contains x/z, deduction fails.

Returns:

- A new enum data type object, if success.
- None, if fail.

Return type: dt object

Examples:

```
>>> from pynpi import waveformw as writer
from collections import OrderedDict
file = writer.create("test_enum.fsdb", "10ns", 10)
file.begin_hierarchy()
file.create_scope(writer.waveform.ScopeType_e.SvModule,"SvModule_sig_enu
m" )
    # creat an enum data type by OrderedDict
    labelDict =
 OrderedDict({"RED":"011","GREEN":None,"BLUE":"101","YELLOW":"","GRAY":"Z
x", "PINK":2 })
    enumDtOD = file.create_enum_dt(labelDict)
    sig_enum_OD = file.create_sig_by_dt_obj(enumDtOD, "sig_enum_OD",
 writer.waveform.DirType_e.DirInput)
    # creat an enum data type by List
    labelList = ["Apple", "Banana", "Cat", "Dog","Zoo"]
    enumDtLI = file.create_enum_dt(labelList)
    sig_enum_LI = file.create_sig_by_dt_obj(enumDtLI, "sig_enum_LI",
 writer.waveform.DirType_e.DirInput)
    file.end_hierarchy()
    sig_enum_OD[0].add_value("GREEN",
 writer.waveform.VctFormat_e.EnumStrVal)
    sig_enum_LI[0].add_value("Zoo",
 writer.waveform.VctFormat_e.EnumStrVal)
    file.incr_time(10)
    sig_enum_OD[0].add_value("3", writer.waveform.VctFormat_e.UintVal)
    sig_enum_LI[0].add_value("2", writer.waveform.VctFormat_e.UintVal)
    file.incr_time(10)
    sig_enum_OD[0].add_value("110",
 writer.waveform.VctFormat_e.BinStrVal)
    sig_enum_LI[0].add_value("100",
 writer.waveform.VctFormat_e.BinStrVal)
    file.incr_time(10)
    sig_enum_OD[0].add_value("Zx", writer.waveform.VctFormat_e.BinStrVal)
    file.incr_time(10)
    sig_enum_OD[0].add_value("10", writer.waveform.VctFormat_e.BinStrVal)
    sig_enum_LI[0].add_value("1", writer.waveform.VctFormat_e.BinStrVal)
    file.incr_time(10)
    writer.close(file)
```

leaf_count(baseDt)

Get the leaf count from a Dt object.

Parameters: **baseDt** – The dt(data type) object.

Returns:

- The leaf count of input data type, if success.
- 0, if fail.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
arrayDt = file.create_array_dt(dt1, 0, 5)
ret = file.leaf_count(arrayDt)
print("Array leaf count ", ret)
Array leaf count 6
```

scope_name(isFull=True)

Get the scope name/fullname of current scope.

Parameters: **isFull** – Specify to get the scope name/fullname.

Returns:

- If isFull is True:
  - • The scope fullname, if success.
- If isFull is False:
  - • The scope name, if success.
- None, if fail.

Return type: str

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test_
1" )
print("Scope full name is", file.scope_name())
file.end_hierarchy()
Scope full name is Module_test.Module_test_1
```

scope_depth()

Get depth of current scope.

Returns: The depth of current scope.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test_
1" )
print("Scope depth is", file.scope_depth())
file.end_hierarchy()
Scope depth is 2
```

create_sig(scalarDtType, name, dir = waveform.DirType_e.DirNone, leftRange = None, rightRange = None)

Create a signal under current scope.

Note:
The signal creation has one limitation on the type of current scope (for example, Verilog signals can only be created under Verilog scope.)

Parameters:

- **scalarDtType** – Scalar Dt type.
- **name** – Signal name.
- **dir** – Direction of the created signal, the default value is waveform.DirType_e.DirNone. (Refer to the PyNPI Waveform Model's enum DirType_e)
- **leftRange** – Left range for vector signal, the default value is None.
- **rightRange** – Right range for vector signal, the default value is None.

Returns:

- Signal Object, if success.
- None, if fail.

Return type:

class waveformw.SigObj(sigObj)

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
sig_basic = file.create_sig(waveformw.ScalarDtType_e.SvLogic,
 "sig_basic", waveformw.waveform.DirType_e.DirInout)
file.end_hierarchy()
```

create_sig_by_dt_obj(dtType, name, dir = waveform.DirType_e.DirNone)

Create a signal under current scope.

Note:
The signal creation has one limitation on the type of current scope (for example, Verilog signals can only be created under Verilog scope.)

Parameters:

dtType – Specific data type.

name – Signal name.

dir – Direction of the created signal, the default value is waveform.DirType_e.DirNone. (Refer to the PyNPI Waveform Model's enum DirType_e)

Returns:

- Signal Object, if success.
- None, if fail.

Return type:

class waveformw.SigObj(sigObj)

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
nameList = ["group_name_1","group_name_2"]
dtList =
 [file.dt(waveformw.ScalarDtType_e.SvLogic),file.dt(waveformw.ScalarDtTyp
e_e.SvInteger) ]
groupDt = file.create_group_dt(ww.GroupType_e.SvStruct, nameList, dtList)
sigDt = file.create_sig_by_dt_obj(groupDt, "sigDt",
 waveformw.waveform.DirType_e.DirInput)
file.end_hierarchy()
```

create_eq_sig(sourceSig, name, dir = waveform.DirType_e.DirNone, dt = None)

Create an equivalent signal. The created signal and source signal shares the same value changes.

Note:
Only same scalar type signal with same size can be created with this API.

Parameters:

sourceSig – Source signal SigObj to create equivalent signal.

name – Signal name.

dir – Direction of the created signal, the default value is waveform.DirType_e.DirNone. (Refer to the PyNPI Waveform Model's enum DirType_e)

dt – Data type of created signal, if dt is None, the created signal has same data type of source signals.

Returns:

- Signal Object, if success.
- None, if fail.

Return type:

class waveformw.SigObj(sigObj)

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
sig_basic = file.create_sig(waveformw.ScalarDtType_e.SvLogic,
 "sig_basic", waveformw.waveform.DirType_e.DirInout)
sig_eq_basic = file.create_eq_sig( sig_basic , "sig_eq_basic")
file.end_hierarchy()
```

Signal

class waveformw.SigObj(*sigObj*)

SigObj Function list:

name()	Get the signal name.
add_value(val, valtype = waveform.VctFormat_e.BinStrVal)	Add a value change at current time for target signal.

Example:

writer.py:

```
import sys, os
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import waveformw as writer
def test():
    # Create the wavform file named "test.fsdb"
    file = writer.create("test.fsdb", "10ns", 10)
    # Begin the hierarchy create phase
```

```
        file.begin_hierarchy()
        # sub-scope create from rootscope
        if
     file.create_scope(writer.waveform.ScopeType_e.SvModule,"SvModule_sig_tes
    t" ) is 1:
            print("Scope full name is", file.scope_name())
        # Up scope to the rootscope
        if file.up_scope() is 0:
            print("Up scope fail.")
        # sub-scope create from rootscope
        if
     file.create_scope(writer.waveform.ScopeType_e.SvModule,"Module_test" )
     is 1:
            print("Scope full name is", file.scope_name())
        # sub-scope create from the scope "Module_test"
        if
     file.create_scope(writer.waveform.ScopeType_e.SvModule,"Module_test_2" )
     is 1:
            print("Scope full name is", file.scope_name())
        # create the sig from the scope "Module_test.Module_test_2"
        sig_basic = file.create_sig(writer.ScalarDtType_e.SvLogic,
     "sig_basic", writer.waveform.DirType_e.DirInout)
        # create an equivalent signal with the source sig
     "Module_test.Module_test_2.sig_basic"
        sig_eq_basic = file.create_eq_sig( sig_basic , "sig_eq_basic")
        # create the sig with range from the scope
     "Module_test.Module_test_2"
        sig_range = file.create_sig(writer.ScalarDtType_e.SvLogic,
     "sig_range", writer.waveform.DirType_e.DirInput, 2, 10)

     file.create_scope(writer.waveform.ScopeType_e.SvModule,"SvModule_name_hi
    er" )
        sig_range_2 = file.create_sig(writer.ScalarDtType_e.SvLogic,
     "sig_range_2", writer.waveform.DirType_e.DirInput, 5, 9)
            print("depth is", file.scope_depth())
            print("Scope full name is", file.scope_name())
            print("Scope name is", file.scope_name(False))
        #create the struct sig by dt
        nameList = ["group_name_1","group_name_2"]
        dtList =
     [file.dt(writer.ScalarDtType_e.SvLogic),file.dt(writer.ScalarDtType_e.Sv
    Integer) ]
        groupDt = file.create_group_dt(writer.GroupType_e.SvStruct, nameList,
     dtList)
        # End the hierarchy create phase
        file.end_hierarchy()
        # Add value to sig
            print("sigDt[1]'s name", sigDt[1].name())
        sigDt[0].add_value("0", writer.waveform.VctFormat_e.BinStrVal)
        sigDt[1].add_value("10", writer.waveform.VctFormat_e.SintVal)
        if sig_basic.add_value("0", writer.waveform.VctFormat_e.BinStrVal) is
     1:
            print("success to add value")
        sig_range.add_value("101110101",
     writer.waveform.VctFormat_e.BinStrVal)
        sig_range_2.add_value("01001", writer.waveform.VctFormat_e.BinStrVal)
            print("sig_basic name is:", sig_basic.name())
        cur_time = file.time()
            print("Time now:", cur_time)
        ret = file.incr_time(15)
        if ret is 1:
            print("Time now:", file.time())
        file.flush()
        sigDt[0].add_value("X")
        sigDt[1].add_value("20", writer.waveform.VctFormat_e.SintVal)
        sig_basic.add_value("1", writer.waveform.VctFormat_e.BinStrVal)
        ret = file.incr_time(15)
        # Create array dt
        dt1 = file.dt(writer.ScalarDtType_e.SvLogic, 3, 10 )
        ret = file.leaf_count(dt1)
            print("Leaf count", ret)
        arrayDt = file.create_array_dt(dt1, 0, 5)
        ret = file.leaf_count(arrayDt)
            print("Array leaf count", ret)
        # Close waveform file
        if writer.close(file) is 1:
            print("Close waveform file.")
        if __name__ == '__main__':
        orig_stdout = sys.stdout
        f = open('writer.log', 'w')
        sys.stdout = f
        test()
        sys.stdout = orig_stdout
        f.close()
```

Result: writer.log

```
And a waveform file named "test.fsdb".
Scope full name is SvModule_sig_test
Scope full name is Module_test
Scope full name is Module_test.Module_test_2
depth is 3
Scope full name is Module_test.Module_test_2.SvModule_name_hier
Scope name is SvModule_name_hier
sigDt[1]'s name sigDt.group_name_2
success to add value
sig_basic name is: sig_basic
Time now: 10
Time now: 25
Leaf count 1
Array leaf count 6
Close waveform file.
```

name()

Get the signal name.

Returns:

- The sig name, if success.
- None, if failed.

Return type: str

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
sig_basic = file.create_sig(waveformw.ScalarDtType_e.SvLogic,
 "sig_basic", waveformw.waveform.DirType_e.DirInout)
file.end_hierarchy()
print("Sig name is ", sig_basic.name() )
Sig name is sig_basic
```

add_value(val, valtype = waveform.VctFormat_e.BinStrVal)

Add a value change at current time for target signal.

Note:
end_hierarchy() should be called before adding value to a signal. Otherwise, this API will fail to add value and return 0.

Parameters:

val – The value of the signal.

valtype – The value type of the value. (Refer to the PyNPI Waveform Model's enum VctFormat_e)

Returns:

- 1, if success.
- 0, if fail.

Return type: int

Examples:

```
>>> file = waveformw.create("test.fsdb", "10ns", 0)
file.begin_hierarchy()
file.create_scope(waveformw.waveform.ScopeType_e.SvModule,"Module_test" )
sig_basic = file.create_sig(waveformw.ScalarDtType_e.SvLogic,
 "sig_basic", waveformw.waveform.DirType_e.DirInout)
file.end_hierarchy()
if sig_basic.add_value("0", waveformw.waveform.VctFormat_e.BinStrVal) is
 1:
```

```
print("Add value success")
Add value success
```