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

Version V-2023.12-SP1, March 2024

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader’s responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

Preface

The Python-Based NPI Transaction 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 seven models. They are as follows:

- Waveform Writer
- Waveform
- Transaction Waveform Writer
- 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 Transaction 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()
```

3

Python-Based NPI Transaction Waveform Writer Model

This chapter includes the following topics:

Abstract

NPI waveform writer model is used to provide efficient and convenient APIs for external users writing their 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 `waveform` to use the APIs of Waveform Writer Model.

   ```
   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']
```

Illegal Cases of Different Handles

To ensure the given handle is valid, most of the APIs checks the handle validation and returns the error message, if invalid. There are some possible reasons for causing the failure.

- Use the handle which is not created by calling the corresponding API. The API for creating the handle is as following:
  1. Transaction FileHandle: `create_tr()`.
  2. Stream Handle: `TrFileHandle.stream_begin()`.
  3. Transaction Handle: `StreamHandle.trans_begin()`.
- Use the handle which is already closed. The API for closing the handle is as following:
  1. Transaction FileHandle: `close()`.
  2. Stream Handle: `StreamHandle.end()`.
  3. Transaction Handle: `TransHandle.end()`.
- For few APIs, the given handles should be closed before used; else, the error is issued. The APIs are as listed in the following:
  1. `StreamHandle.trans_begin()`: The given stream handle should be closed(`StreamHandle.end()`) in advance.
  2. `TransHandle.add_relation()`: The given transaction handles - master (self) and slave should be closed (`TransHandle.end()`) in advance.

Data Types of Attributes

This section summarizes the Verdi transaction data type categories and their definitions including Transaction, Stream, Attribute, and Transaction Relation.

- **Transaction**: Transaction is a one entry of data input by user code (e.g. an UVM message) and it contains attributes of different data types. For maximum flexibility, transactions can be in different types. The predefined transaction types are as following:

  1. `TransType_e.Message`: Regular transactions for printf-like logging purpose. The transaction in this type will be set as a 0-time transaction automatically, which means the end time of this transaction is set equal to the begin time of the transaction.
  2. `TransType_e.Action`: A zero time transaction for recording few Actions (e.g. read, write, interrupt). The transaction whose type is Action will be set as zero time automatically. Several Action transactions can be formed into a Group transaction by adding the `Rel_e.BelongTo` between the Action transaction and the Group transaction.
  3. `TransType_e.Transaction`: A general transaction with a begin time and an end time. Transaction can be set as zero time manually by setting the end time and the begin time with same value.
  4. `TransType_e.Group`: A Group contains several transactions (usually of Action type). The begin time of this transaction will be set as the begin time of its first member transaction. Also, the end time of this transaction will be set as the end time of its last member transaction.

- **Stream**: A Stream is used to organize (or record as a Group in the FSDB file) a temporal sequence of transactions grouped by some criteria, for example, transactions originating from the same UVM component can be put into one stream. Different streams cannot share the same transaction. Some streams have hierarchies (e.g. following the UVM component hierarchy), some do not (e.g. UVM phases, objections). Streams may form different hierarchies (e.g. UVM component hierarchy and sequence hierarchy).

- **Attribute**: An Attribute is a characteristic or property of a specific scope, stream, or transaction.

- **Transaction Relation**: A Relation is a link between two transactions. It can be defined using the API `TransHandle.add_relation()`. The parameter relation can be as following:

  1. `Rel_e.BelongTo`: It indicates the belong relation between two transactions, that is, first transaction Master belongs to second transaction Slave. The transaction Slave can only be in `TransType_e.Group` type and the transaction Master cannot be in `TransType_e.Group` type. In practical terms, Master is always in `TransType_e.Action` type and Slave has several transactions belonging to it.

2. `Rel_e.AnnotateTo`: It indicates the annotative relation between two transactions, that is, first transaction Master (annotator) annotates to second transaction Slave (annotate). Master can only be in `TransType_e.Message` type.
3. `Rel_e.ParentChild`: It indicates the layering relation between two transactions, that is, first transaction Master is a parent and second transaction Slave is a child. This relation is valid only if both Master and Slave are in `TransType_e.Transaction` type. Master (parent) and Slave (child) in different streams are allowable for this relation.
4. `Rel_e.PredSucc`: It indicates the ordering/triggering relation between two transactions, that is, first transaction Master is a predecessor and transaction Slave is a successor. The begin time of the transaction Master should be less than the begin time of the transaction Slave.
5. User defined string: It means the user can define the relation between two transactions. The given rel should be in the type of string.

- **Legal Transaction Type Table**:

Relation type	**Master**	**Slave**
`Rel_e.BelongTo`	Any Type except `TransType_e.Group`	`TransType_e.Group`
`Rel_e.AnnotateTo`	`TransType_e.Message`	Any Type
`Rel_e.ParentChild`	`TransType_e.Transaction`	`TransType_e.Transaction`
`Rel_e.PredSucc`	Any Type	Any Type
User defined string	Any Type	Any Type

Enums

- Enum list
- Radix Enum
  - Radix_e: Transaction waveform writer attribute radix.
- Transaction Type Enum
  - TransType_e: Transaction waveform writer transaction type.
- Relation Type Enum
  - Rel_e: Transaction waveform writer transaction relation type.

- Attribute Type Enum
  - AttrType_e: Transaction waveform writer transaction attribute value type.

Radix Enum

Radix_e

class waveformw.**Radix_e**

Transaction waveform writer attribute radix. `Radix_e` is used to identify the attribute radix of add-attribute APIs.

Bin: Representing the binary system for attribute value.

Oct: Representing the octal system for attribute value.

Dec: Representing the decimal system for attribute value.

Hex: Representing the hexadecimal system for attribute value.

Unsigned: Representing unsigned for attribute value.

Bin = 0

Oct = 1

Dec = 2

Hex = 3

Unsigned = 4

Transaction Type Enum

TransType_e

class waveformw.**TransType_e**

Transaction waveform writer transaction type. `TransType_e` is used to identify the predefined transaction type.

Message: Regular transactions for printf-like logging purpose. The transaction in this type will be set as a zero time transaction automatically, which means the end time of this transaction is set equal to the begin time of the transaction.

Action: A zero time transaction for recording some Actions (e.g. read, write, interrupt). The transaction whose type is action will be set as zero time automatically. Several Action transactions can be formed into a Group transaction by adding the `Belong_To` relation between the Action transaction and the Group transaction.

Transaction: A general transaction with a begin time and an end time. Transaction can be set as zero time manually by setting the end time and the begin time in same value.

Group: A group contains several transactions (usually of Action type). The begin time of this transaction will be set as the begin time of its first member transaction. Also, the end time of this transaction will be set as the end time of its last member transaction.

Message = 0

Action = 1

Transaction = 2

Group = 3

Relation Type Enum

Rel_e

class waveformw.**Rel_e**

Transaction waveform writer transaction relation type. `Rel_e` is used to identify the transaction relation type.

BelongTo: It indicates the belong relation between two transactions, that is, first transaction Master belongs to second transaction Slave. The transaction Slave can only be in group type and the transaction Master cannot be in the group type.

In practical terms, Master is always in Action type and Slave has several transactions belonging to it.

AnnotateTo: It indicates the annotative relation between two transactions, that is, first transaction Master (annotator) annotates to second transaction Slave (annotate). Master can only be in message type.

ParentChild: It indicates the layering relation between two transactions, that is, first transaction Master is a parent and transaction Slave is a child. This relation is valid only if both Master and Slave are in the transaction type. Master (parent) and Slave (child) in different streams is allowable for this relation.

PredSucc: It indicates the ordering/triggering relation between two transactions, that is, first transaction Master is a predecessor and transaction Slave is a successor. The begin time of the transaction Master should be less than the begin time of the transaction Slave.

BelongTo = 0

AnnotateTo = 1

ParentChild = 2

PredSucc = 3

Attribute Type Enum

AttrType_e

class waveformw.**AttrType_e**

Transaction waveform writer transaction attribute value type. `AttrType_e` is used to identify the transaction attribute value type.

Float: It represents floating point type. The size is 32 bits.

Double: It represents floating point type. The size is 64 bits.

String: It represents string type.

Char: It represents char type. The size is 8 bits.

Short: It represents short integer type. The size is 16 bits.

Int: It represents integer type. The size is 32 bits.

LongLong: It represents long long integer type. The size is 64 bits.

BitVec: It represents bit vector.

Float = 0

Double = 1

String = 2

Char = 3

Short = 4

Int = 5

LongLong = 6

BitVec = 7

Feedback

L0 APIs

- Transaction File
- Stream
- Transaction
- AttrFactory

Transaction File

Function list

create_tr([name, unit, beginTime])	Create a Transaction waveform file with a given file name in the current working directory.
close(file)	Close the Waveform file.

Example:

Following is an example showing how to create a transaction 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 npisys
from pynpi import waveformw as writer

npisys.init(sys.argv)

# Create a waveform file "novas.fsdb"
file = writer.create_tr("novas.fsdb", "10ns", 10)

# close the waveform file
writer.close(file)
npisys.end()
```

Result:

You will get a transaction waveform file named `novas.fsdb` in your current `dir`.

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

Create a Transaction 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 set as [1, 10, 100][s, ms, us, ns, ps, fs]. It will be set as 1 ns, if other value is given.
- **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:

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

Return Type: TrFileHandle

Examples:

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

waveformw.close(*file*)

Close the Waveform file.

Parameters: **file** – The Waveform file Object FileObj - Refer to the *Python-Based NPI Waveform Writer Model User Guide* or the Transaction Waveform file object TrFileHandle to be closed.

Returns:

- 1, if success.
- 0, if fail.
- None, if the file is not a Waveform file Object FileObj - Refer to the *Python-Based NPI Waveform Writer Model User Guide* nor a Transaction Waveform file object TrFileHandle.

Return type: int

Examples:

```
>>> print("Close file ret is ", waveformw.close(file))
Close file ret is 1
```

Feedback

class waveformw.TrFileHandle(tr*FileObj*)

FileObj Function list:

name()	Get name of the Transaction waveform file object.
scale_unit()	Get scale unit of the Transaction waveform file object.
flush()	Flush the data into Transaction waveform file obj without closing it.
incr_time(time)	Step forward with the given time value in simulation time.
time()	Get the current time recorded by the specific Transaction waveform file object.
scope_add_attr(scopeName, attrDict[, checkAttr])	Add an attribute to a scope.
stream_begin(name)	Create a stream with a given path.

Example:

trFile.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
from pynpi import npisys

def test():
    # Create the wavform file named "test_tr.fsdb"
    file = writer.create_tr("test_tr.fsdb", "1ns" , 10)

    if file == None:
      return 0
    print( "New tr wave file: ", file.name(), " scale unit: " ,
 file.scale_unit() , " begin time:" , file.time() )
    if 1 == file.flush():
      print( " File flush success. " )

    if 1 == file.incr_time(10):
      print( "After incr time, curr time is :" , file.time() )

    # attrFactory
dictUserDefinedChk = { 'name' : 'afDictUserDefined', 'val' :
 'afDictUserDefinedVal', 'type' : writer.AttrType_e.String, 'radix' :
 writer.Radix_e.Dec }
```

```
dictUserDefinedNchk = { 'name' : 'afDictUserDefinedN', 'val' :
 'afDictUserDefinedVal', 'type' : writer.AttrType_e.String, 'radix' :
 writer.Radix_e.Dec }

    afTest = writer.AttrFactory(afDict = dictUserDefinedChk)

    scopeName = "trTop.scope.dictUserDefinedChk"
    if 1 == file.scope_add_attr(scopeName , afTest.to_dict() ):
        print( "Add scope ", scopeName , " success." )

    scopeName = "trTop.scope.dictUserDefinedNchk"
    if 1 == file.scope_add_attr(scopeName , dictUserDefinedNchk, False):
        print( "Add scope ", scopeName , " success." )

    file.incr_time(20)
    afDefault = writer.AttrFactory()
    # add scope with attribute
    scopeName = "trTop.scope.dictAfDefaultAll"
    afDefault.set_name('afDefaultFloat')
    afDefault.set_attr(11.111111, writer.AttrType_e.Float)
    afDefaultFromAF = afDefault.to_dict()
    if 1 == file.scope_add_attr(scopeName , afDefaultFromAF ):
        print( "Add scope type", afDefaultFromAF["name"] , " success." )

    afDefault.set_name('afDefaultDouble')
    afDefault.set_attr(12.121212, writer.AttrType_e.Double)
    afDefaultFromAF = afDefault.to_dict()
    if file.scope_add_attr(scopeName , afDefaultFromAF ):
        print( "Add scope type", afDefaultFromAF["name"] , " success." )

    afDefault.set_name('afDefaultString')
    afDefault.set_attr("testDtring", writer.AttrType_e.String)
    afDefaultFromAF = afDefault.to_dict()
    if file.scope_add_attr(scopeName , afDefaultFromAF ):
        print( "Add scope type", afDefaultFromAF["name"] , " success." )

    afDefault.set_name('afDefaultChar')
    afDefault.set_attr("a", writer.AttrType_e.Char)
    afDefaultFromAF = afDefault.to_dict()
    if afDefaultFromAF and file.scope_add_attr(scopeName ,
 afDefaultFromAF ):
        print( "Add scope type", afDefaultFromAF["name"] , " success." )
    afDefault.set_name('afDefaultShort')
    afDefault.set_attr(13, writer.AttrType_e.Short)
    afDefaultFromAF = afDefault.to_dict()
    if afDefaultFromAF and file.scope_add_attr(scopeName ,
 afDefaultFromAF ):
        print( "Add scope type", afDefaultFromAF["name"] , " success." )

    afDefault.set_name('afDefaultInt')
    afDefault.set_attr(133, writer.AttrType_e.Int)
    afDefaultFromAF = afDefault.to_dict()
```

```
   if afDefaultFromAF and file.scope_add_attr(scopeName ,
afDefaultFromAF ):
       print( "Add scope type", afDefaultFromAF["name"] , " success." )

   afDefault.set_name('afDefaultLongLong')
   afDefault.set_attr(1414141414, writer.AttrType_e.LongLong)
   afDefaultFromAF = afDefault.to_dict()
   if afDefaultFromAF and file.scope_add_attr(scopeName ,
afDefaultFromAF ):
       print( "Add scope type", afDefaultFromAF["name"] , " success." )

   # create stream
   stream = file.stream_begin("scope1.scope2.scope3.stream1")
   if stream:
       print("stream creat with name: ", stream.name() , " full is: ",
stream.full_name())
   stream.end()
   writer.close(file)

   if __name__ == '__main__':
   orig_stdout = sys.stdout
   f = open('writer_tr.log', 'w' )
   sys.stdout = f
   npisys.init(sys.argv)
   test()
   npisys.end()
   sys.stdout = orig_stdout
   f.close()
```

Result: writer.log

And a waveform file named `test_tr.fsdb`

```
New tr wave file: ./myFolder/test_tr.fsdb scale unit: 1ns begin time: 10
File flush success.
After incr time, curr time is : 20
Add scope trTop.scope.dictUserDefinedChk success.
Add scope trTop.scope.dictUserDefinedNchk success.
Add scope type afDefaultFloat success.
Add scope type afDefaultDouble success.
Add scope type afDefaultString success
Add scope type afDefaultChar success.
Add scope type afDefaultShort success.
Add scope type afDefaultInt success.
Add scope type afDefaultLongLong success.
stream creat with name: stream1 full is:
 $trans_root.scope1.scope2.scope3.stream1
```

name()

Get name of the Transaction waveform file object.

Returns:

- File name, if success.
- None, if fail.

Return type: str

Examples:

```
>>> print(file.name())
./myFolder/novas.fsdb
```

scale_unit()

Get scale unit of the Transaction waveform file object.

Returns:

- Scale unit, if success.
- None, if fail.

Return type: str

Examples:

```
>>> print(file.scale_unit())
10ns
```

flush()

Flush the data into Transaction waveform file obj without closing it.

Returns:

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

Return type: int

Examples:

```
>>> trFile = waveformw.create_tr("test.fsdb", "10ns", 10)
trFile.flush()
```

incr_time(time)

Step forward with the given time value in simulation time.

Args:

time: Target increase time in the specified fie handle.If it causes the current time overflow, an error will be issued and this API is failed.

Returns:

```
1, if success.
0, if fail.
```

Return type: int

Examples:

```
>>> trFile = waveformw.create_tr("test.fsdb", "10ns", 10)
trFile.incr_time(30)
ptint(trFile.time())
40
```

time()

Get the current time recorded by the specific Transaction waveform file object.

Returns:

```
Current time, if success.
None, if fail.
```

Return type: int

Examples:

```
>>> trFile = waveformw.create_tr("test.fsdb", "10ns", 10)
ptint(trFile.time())
10
```

scope_add_attr(scopeName, attrDict, checkAttr=True)

Add an attribute to a scope.

Parameters:

scopeName –

- The full hierarchy name of the scope, use '.' as delimiter.
- If it is None, an error will be issued and this API is failed.

attrDict –

A dictionary with attribute information.

The dictionary spec shown as following:

```
{ "name" : "attrName", \ # The name of attribute.
"val" : "attrValue", \ # Attribute value with given Attribute type
"type" : writer.AttrType_e.String, \ # Attribute value's
 type AttrType_e . AttrType_e.BitVec is not support for stream attribute.
"radix" : writer.Radix_e.Dec \ # Optional key, default set
 as Radix_e.Dec . The radix Radix_e of the integer data type being used.
}
```

checkAttr –

Enable the option to check the attrDict is in spec. This option is default on.

Returns:

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

Return type: int

Examples:

```
>>> file.scope_add_attr("trTop.scope" , attrDict )
```

stream_begin(name)

Create a stream with a given path.

Parameters:

name – Full path name(delimiter is ‘.’).

Returns:

- StreamHandle object, if success.
- None, if fail.

Return type: StreamHandle

Examples:

```
>>> file.stream_begin("scope1.scope2.scope3.stream1")
```

Stream

class waveformw.StreamHandle(*streamObj*)

StreamHandle Function list

name()	Get name of stream object.
full_name()	Get the full name of stream object.
define_attr(name[, attrType, attrRadix])	Define a dense attribute on a specific stream.
add_attr(attrDict[, checkAttr])	Add an attribute to a stream.
end()	End the stream creation.
trans_begin([type, beginTime])	Create a transaction with a specified transaction type.

Examples:

stream.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
from pynpi import npisys
def test():
    # Create the wavform file named "test_tr.fsdb"
    file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
    stream = file.stream_begin("scope1.scope2.scope3.stream1")
    if not stream:
         writer.close(file)
         return 0
    print("stream creat with name: ", stream.name() , " full is: ",
 stream.full_name())
    # define attr
    if stream.define_attr('streamDefineAttrFloat',
 writer.AttrType_e.Float):
        print("stream define attr Float success.")
    if stream.define_attr('streamDefineAttrDouble',
 writer.AttrType_e.Double):
        print("stream define attr Double success.")
    if stream.define_attr('mDefineAttrString',
 writer.AttrType_e.String ):
        print("stream define attr String success.")
    if stream.define_attr('streamDefineAttrChar',
 writer.AttrType_e.Char):
        print("stream define attr Char success.")
    if stream.define_attr('streamDefineAttrShort',
 writer.AttrType_e.Short, writer.Radix_e.Bin):
        print("stream define attr Short success.")
    if stream.define_attr('streamDefineAttrInt', writer.AttrType_e.Int,
 writer.Radix_e.Hex):
        print("stream define attr Int success.")
    if stream.define_attr('streamDefineAttrLongLong',
 writer.AttrType_e.LongLong):
        print("stream define attr LongLong success.")
    # stream add attribute
    if stream.add_attr(writer.AttrFactory('streamAddAttrFloat', 31.31,
 writer.AttrType_e.Float).to_dict()):
        print("stream add attr Float success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrDouble',
 32.32323232, writer.AttrType_e.Double).to_dict()):
        print("stream add attr Double success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrString', "Test",
 writer.AttrType_e.String).to_dict()):
        print("stream add attr String success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrChar', "c",
 writer.AttrType_e.Char).to_dict()):
        print("stream add attr Char success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrShort', 33,
 writer.AttrType_e.Short).to_dict()):
        print("stream add attr Short success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrInt', 333,
 writer.AttrType_e.Int).to_dict()):
        print("stream add attr Int success.")
    if stream.add_attr(writer.AttrFactory('streamAddAttrLongLong',
 3434343434, writer.AttrType_e.LongLong).to_dict()):
        print("stream add attr LongLong success.")
    stream.end()
    # create transaction
    trans1 = stream.trans_begin( beginTime = 5 )
    if trans1:
        print("trans begin success.")
    trans1.end()
    writer.close(file)
    if __name__ == '__main__':
    orig_stdout = sys.stdout
    f = open('writer_stream.log', 'w' )
    sys.stdout = f
    npisys.init(sys.argv)
    test()
    npisys.end()
    sys.stdout = orig_stdout
    f.close()
```

Result:

writer_stream.log

And a waveform file named `test_tr.fsdb`.

```
stream creat with name: stream1 full is:
 $trans_root.scope1.scope2.scope3.stream1
stream define attr Float success.
stream define attr Double success.
stream define attr String success.
stream define attr Char success.
stream define attr Short success.
stream define attr Int success.
stream define attr LongLong success.
stream add attr Float success.
stream add attr Double success.
stream add attr Char success.
stream add attr Short success.
stream add attr Int success.
stream add attr LongLong success.
trans begin success.
```

name()

Get name of stream object.

Returns:

- The stream name, if success.
- None, if fail.

Return type: str

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
if stream:
print("stream creat with name: ", stream.name() , " full is:
", stream.full_name())
stream.end()
stream creat with name: stream1 full is:
$trans_root.scope1.scope2.scope3.stream1
```

full_name()

Get full name of the stream object.

Returns:

- The full name of stream object, if success.
- None, if fail.

Return type: str

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
if stream:
print("stream creat with name: ", stream.name() , " full is: ",
 stream.full_name())
stream.end()
stream creat with name: stream1 full is:
 $trans_root.scope1.scope2.scope3.stream1
```

define_attr(name, attrType=<AttrType_e.String: 2>, attrRadix=<Radix_e.Dec: 2>)

Define a dense attribute on a specific stream. All transactions in this stream will have this attribute.

Parameters:

name – The name of attribute.

attrType – Attribute value's type AttrType_e. Default value is `AttrType_e.String`. `AttrType_e.BitVec` will not support this function.

attrRadix – The radix Radix_e of the integer data type being used. Default value is `Radix_e.Dec`.

Returns:

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

Return type: int

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
if stream:
stream.define_attr('streamDefineAttrShort', writer.AttrType_e.Short,
 writer.Radix_e.Bin)
```

add_attr(attrDict, checkAttr=True)

Add an attribute to a stream.

Parameters:

```
attrDict - A dictionary with attribute information. The dictionary spec
 shown as following:
{ "name" : "attrName", \ # The name of attribute.
"val" : "attrValue", \ # Attribute value with given Attribute type
"type" : writer.AttrType_e.String, \ # Attribute value's
 type AttrType_e . AttrType_e.BitVec is not support for stream attribute.
"radix" : writer.Radix_e.Dec \ # Optional key, default set
 as Radix_e.Dec . The radix Radix_e of the integer data type being used.
}
```

checkAttr – Enable the option to check the attrDict is in spec. This option is default on.

Returns:

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

Return type: int

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
if stream:
stream.add_attr(writer.AttrFactory('streamAddAttrShort', 33,
 writer.AttrType_e.Short).to_dict())
```

end()

End the stream creation.

Note:

A stream is completely created after calling this API.

Returns:

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

Return type: int

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
if stream:
stream.add_attr(writer.AttrFactory('streamAddAttrShort', 33,
 writer.AttrType_e.Short).to_dict())
stream.end()
```

trans_begin(type=<TransType_e.Transaction: 2>, beginTime=-1)

Create a transaction with a specified transaction type. The begin time of this transaction is set as the current time.

Parameters:

type – TransType_e.

beginTime – The Transaction begin time. If the begin time of transaction is not given, it will be set to the current simulation time.

The input time should not be greater than the current time; else, an error will be issued and `beginTime` is set to the current time.

Returns:

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

Return type: TransHandle

Examples:

```
>>> file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
stream = file.stream_begin("scope1.scope2.scope3.stream1")
stream.end()
trans1 = stream.trans_begin( beginTime = 5 )
if trans1:
print("trans begin success.")
trans1.end()
trans begin success.
```

Transaction

class waveformw.TransHandle(*transObj*)

TransHandle Function list

set_label([label])	Specify the transaction label.
add_tag(tag)	Specify the transaction tag.
add_attr(attrDict[, checkAttr])	Add an attribute into a specific transaction with/without a given expected attribute.
end()	End a transaction.
add_relation(rel, slaveTrans)	This is to specify the relation between two different transactions.

Examples:

trt.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
from pynpi import npisys
def test():
    # Create the wavform file named "test_tr.fsdb"
    file = writer.create_tr("test_tr.fsdb", "1ns" , 10)
    file.incr_time(20)
    stream = file.stream_begin("scope1.scope2.scope3.stream1")
    if stream.define_attr('streamDefineAttrFloat',
 writer.AttrType_e.Float):
        print("stream define attr Float success.")
    stream.end()
    file.incr_time(30)
    # create transaction
    label = ""
    trans1 = stream.trans_begin( beginTime = 5 )
    if not trans1:
        print("trans1 begin fail.")
    else:
    if trans1.set_label():
        print("trans1 set label without input success.")
    # transaction add attribute
    # add stream defined attribute
    if trans1.add_attr(writer.AttrFactory('streamDefineAttrFloat', 48.48,
 writer.AttrType_e.Float).to_dict()):
        print("tran1 add attr Float Define success.")
    # transaction add regular attribute
    if trans1.add_attr(writer.AttrFactory('tranAddAttrFloat', 41.41,
 writer.AttrType_e.Float).to_dict()):
        print("tran1 add attr Float success.")
    if trans1.add_attr(writer.AttrFactory('tranAddAttrInt', 433,
 writer.AttrType_e.Int).to_dict()):
        print("tran1 add attr Int success.")
    if trans1.add_attr(writer.AttrFactory('tranAddAttrBitVec',
 "00001000", writer.AttrType_e.BitVec).to_dict()):
        print("tran1 add attr BitVec success.")
    trans1.end()
    file.incr_time(30)
    trans2 = stream.trans_begin( beginTime = 70 )
    label = "label2"
    tag = "tag2"
    if not trans2:
        print("trans2 begin fail.")
    else:
    if trans2.set_label("lable2"):
        print("trans2 set label.")
    if trans2.add_tag("tag2"):
        print("trans2 add_tag success.")
    # transaction add attribute with expected value
    if trans2.add_attr(writer.AttrFactory('tranAddAttrDouble',
 52.52525252, writer.AttrType_e.Double , 62.62626262).to_dict()):
        print("tran2 add attr Double success.")
    if trans2.add_attr(writer.AttrFactory('tranAddAttrString',
 "tranAddAttrExp", writer.AttrType_e.String,
 "tranAddAttrExpVal").to_dict()):
        print("tran2 add attr String success.")
    if trans2.add_attr(writer.AttrFactory('tranAddAttrChar', "e",
 writer.AttrType_e.Char, "f").to_dict()):
        print("tran2 add attr Char success.")
    if trans2.add_attr(writer.AttrFactory('tranAddAttrLongLong',
 5454545454, writer.AttrType_e.LongLong, 6464646464).to_dict()):
        print("tran2 add attr LongLong success.")
    if trans2.add_attr(writer.AttrFactory('tranAddAttrBitVec',
 "01010101", writer.AttrType_e.BitVec, "10101010").to_dict()):
        print("tran2 add attr BitVec success.")
    trans2.end()
    # add relation with Rel_2 type
    if trans1.add_relation( writer.Rel_e.ParentChild , trans2):
        print("add relation success.")
    # add relation with user define string
    if trans2.add_relation( "test_rel", trans1):
        print("add relation self defined success.")
    writer.close(file)
    if __name__ == '__main__':
    orig_stdout = sys.stdout
    f = open('writer_trt.log', 'w' )
    sys.stdout = f
    npisys.init(sys.argv)
    test()
    npisys.end()
```

Result:

writer_trt.log

And a waveform file named `test_tr.fsdb`.

```
stream define attr Float success.
trans1 set label without input success.
tran1 add attr Float Define success.
tran1 add attr Float success.
tran1 add attr Int success.
tran1 add attr BitVec success.
trans2 set label.
trans2 add_tag success.
tran2 add attr Double success.
tran2 add attr String success.
tran2 add attr Char success.
tran2 add attr LongLong success.
tran2 add attr BitVec success.
```

```
add relation success.
add relation self defined success.
```

set_label(label='')

Specify the transaction label.

Parameters:

label – The label of this transaction. If the API is called multiple times, the last one will take effect. If it is omitted, it is set as empty string. If it is None, an error will be issued and this API is failed.

Returns:

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

Return type: int

Examples:

```
>>> stream = file.stream_begin("scope1.scope2.scope3.stream1")
stream.end()
trans1 = stream.trans_begin( beginTime = 5 )
trans1.set_label("test_label")
trans1.end()
```

add_tag(tag)

Specify the transaction tag.

Parameters:

tag – The specified tag in string type. If this application is called multiple times, all the tags are recorded.

Returns:

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

Return type: int

Examples:

```
>>> stream = file.stream_begin("scope1.scope2.scope3.stream1")
stream.end()
trans1 = stream.trans_begin( beginTime = 5 )
trans1.add_tag("test_tag")
trans1.end()
```

end()

End a transaction.

Returns:

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

Return type: int

Examples:

```
>>> stream = file.stream_begin("scope1.scope2.scope3.stream1")
stream.end()
trans1 = stream.trans_begin( beginTime = 5 )
trans1.end()
```

add_attr(attrDict, checkAttr=True)

Add an attribute into a specific transaction with or without a given expected attribute.

Parameters:

attrDict – A dictionary with attribute information. The dictionary spec shown as following:

```
{ "name" : "attrName", \ # The name of attribute.
"val" : "attrValue", \ # Attribute value with given Attribute type.
"type" : writer.AttrType_e.String, \ # Attribute value's
 type AttrType_e .
"expectVal" : "attrExpectValue", \ # Expect Attribute value with given
 Attribute type.
"radix" : writer.Radix_e.Dec \ # Optional key, default set
 as Radix_e.Dec. The radix Radix_e of the integer data type being used.
}
```

checkAttr – Enable the option to check the `attrDict` is in spec. This option is default on.

Returns:

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

Return type: int

Examples:

```
>>> stream = file.stream_begin("scope1.scope2.scope3.stream1")
stream.end()
trans1 = stream.trans_begin( beginTime = 5 )
```

```
trans1.add_attr(writer.AttrFactory('tranAddAttrFloat', 41.41,
 writer.AttrType_e.Float).to_dict())
trans1.end()
```

add_relation(rel, slaveTrans)

This is to specify the relation between two different transactions.

Parameters:

rel – Rel_e or the string type relation name.

slaveTrans – The slave TransHandle to specify the relation.

Returns:

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

Return type: int

Examples:

```
>>> stream = file.stream_begin("scope1.scope2.scope3.stream1")
trans1 = stream.trans_begin( beginTime = 5 )
trans1.end()
trans2 = stream.trans_begin( beginTime = 70 )
trans2.end()
trans1.add_relation( writer.Rel_e.ParentChild , trans2)
trans2.add_relation( "test_rel", trans1)
```

AttrFactory

class waveformw.AttrFactory(*name=None, val=None, type=None, expectVal=None, radix=None, afDict=None*)

`AttrFactory` helps users to create the attribute dictionary in spec.

Parameters:

name – Name of a attribute. Default value is None.

val – Attribute Value. Default value is None.

type – The AttrType_e type of Attribute Value and Expect Attribute Value. Default value is AttrType_e.String.

expectVal – Expect Attribute Value. Default value is None.

radix – The radix Radix_e of the integer data type being used. Default value is Radix_e.Dec.

afDict – The attribute info with dictionary type. Default value is None.

AttrFactory Function list

set_name(name)	Set attribute name.
set_attr(val[, type, expectVal])	Set attribute value, attribute type and expect attribute value.
set_expect_attr(expectVal)	Set expect attribute value.
set_radix(radix)	Set attribute radix.
reset()	Reset attribute information.
to_dict([checkAttr])	Construct *AttrFactory* to a dictionary with spectify key and user given information.

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
from pynpi import npisys
npisys.init(sys.argv)
print( writer.AttrFactory('streamAddAttrInt', 333,
 writer.AttrType_e.Int).to_dict(True) )
afDefault = writer.AttrFactory()
afDefault.set_name('afDefaultLongLong')
afDefault.set_attr(1414141414, writer.AttrType_e.LongLong)
print( afDefault.to_dict() )
npisys.end()
```

Result:

```
{'name': 'streamAddAttrInt', 'val': 333, 'type': <AttrType_e.Int: 5>,
'expectVal': None, 'radix': <Radix_e.Dec: 2>}
{'name': 'afDefaultLongLong', 'val': 1414141414, 'type':
<AttrType_e.LongLong: 6>, 'expectVal': None, 'radix': <Radix_e.Dec:
2>}
```

set_name(name)

Set attribute name.

Parameters:

name – Name of an attribute.

Examples:

```
>>> afDefault = writer.AttrFactory()
afDefault.set_name('afSetName')
```

set_attr(val, type=<AttrType_e.String: 2>, expectVal=None)

Set attribute value, attribute type and expect attribute value.

Parameters:

val – Attribute value with given Attribute type.

type – The AttrType_e type of Attribute Value and Expect Attribute Value. Default value is AttrType_e.String.

expectVal – Expect attribute value with given Attribute type.

Examples:

```
>>> afDefault = writer.AttrFactory()
afDefault.set_attr(11.111111, writer.AttrType_e.Float)
```

set_expect_attr(expectVal)

Set expect attribute value.

Parameters:

expectVal – Expect attribute value with given Attribute type.

Examples:

```
>>> afDefault = writer.AttrFactory()
afDefault.set_expect_attr(23)
```

set_radix(radix)

Set attribute radix.

Parameters:

radix – The radix Radix_e of the integer data type being used.

Examples:

```
>>> afDefault = writer.AttrFactory()
afDefault.set_radix(writer.Radix_e.Dec)
```

reset()

Reset attribute information.

Examples:

```
>>> afDefault = writer.AttrFactory()
afDefault.set_attr(11.111111, writer.AttrType_e.Float, 23.23)
afDefault.reset()
```

to_dict(checkAttr=False)

Construct `AttrFactory` to a dictionary with spectify key and user given information.

Parameters:

checkAttr – Enable the option to check the output attribute dictionary is in spec. This option is default off.

Returns:

A dictionary with spectify key and user given information.

Return type: Dict

Examples:

```
>>> print( writer.AttrFactory('streamAddAttrInt', 333,
writer.AttrType_e.Int).to_dict(True) )
{'name': 'streamAddAttrInt', 'val': 333, 'type': <AttrType_e.Int: 5>,
'expectVal': None, 'radix': <Radix_e.Dec: 2>}
```