Verification Continuum™ Verdi®
Python-Based NPI Text Model

Version V-2023.12-SP1, March 2024

SYNOPSYS®

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

Feedback

Contents

Preface

The Python-Based NPI Text Model User Guide helps you to traverse or manipulate design source codes from a text perspective.

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:

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

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 Text** is demonstrated.

Packages and Modules

This section describes the packages and modules.

Packages

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

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

This section describes the 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.

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. Ensure that you set it before running the program.

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

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

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

4. Import “npisys” module 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, to import text module, use the following command:

```
from pynpi import text
```

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

   Python program to use NPI models: (`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 text
# 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 example to setup environment and execute Python program:

```
(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

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.

Feedback

Return type: int

Example

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

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 Text Model

This chapter includes the following topics:

Overview

The NPI Text Model helps you to traverse or manipulate design source codes from a text perspective. Every file, line, and word is regarded as an object and the traversing and manipulative APIs are provided.

In addition to the basic insert, delete, and replace capabilities, you can also access the attribute of each word from the design analysis results of the Verdi system.

Quick Start

This section describes the design, example code, and result.

Design (inc.v)

```
`define TOP(a,b) a + b - 1
module top (input clk);
initial
$display( `TOP( `TOP(2,3), `TOP(22,33) ) );
endmodule
```

Example Code:

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

Feedback

```
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import npisys
from pynpi import text
# -- init ------------------------------------------------------------
res = npisys.init(sys.argv)
if res != 1:
    print('[Error] Failed to do npi init')
    sys.exit(1)
# -- load design -----------------------------------------------------
res = npisys.load_design(sys.argv)
    if res != 1:
    print('[Error] Failed to do npi load design')
    sys.exit(1)
# -- get file list ---------------------------------------------------
textFileList = text.get_file_list()
if len(textFileList) <= 0:
    print('[Error] Failed to get file hdl')
    sys.exit(1)
# -- traverse all files and words and expand macro and show macro
 information after $display ------
for file in textFileList:
    lineList = file.line_handles()
    for line in lineList:
      if (line.line_content().find('$display') > 0):
          print(line.line_content())
          wordList = line.word_handles()
          for word in wordList:
              if (word.word_attribute() == 'npiTextMacroName'):
              macro = word.text_macro()
              print("** macro name: %s" %(word.word_name()));
              print("** macro is ok: %d" %(macro.ok()))
              print("** macro arg num: %d" %(macro.arg_num()))
              print("** macro value: %s" %(macro.value()))
              for i in range(macro.arg_num()):
                print("** macro arg (%d): %s" %(i, macro.arg(i)))
                print("** macro def file: %s" %(macro.def_file()))
                print("** macro def line: %d" %(macro.def_line()))
                print("** macro def value: %s" %(macro.def_value()))
              for i in range(macro.arg_num()):
                print("** macro def arg (%d): %s" %(i, macro.def_arg(i)))
              if (word.expand_macro() == 1):
                print(' ===> expand macro test ok!')
                print(line.line_content())
# -- end -------------------------------------------------------------
npisys.end()
```

Execution Result

```
$display( `TOP( `TOP(2,3), `TOP(22,33) ) );
** macro name: TOP
** macro is ok: 1
```

Feedback

```
** macro arg num: 2
** macro value: 2 + 3 - 1 + 22 + 33 - 1 - 1
** macro arg (0): `TOP(2,3)
** macro arg (1): `TOP(22,33)
** macro def file: inc.v
** macro def line: 1
** macro def value: a + b - 1
** macro def arg (0): a
** macro def arg (1): b
===> expand macro test ok!
$display( 2 + 3 - 1 + 22 + 33 - 1 - 1 );
```

Class Objects

In text module, all the data are composed of three class objects: “File”, “Line” and “Word”. Users can use L0 APIs to get “File” objects and get their “Line” objects their class object methods. Then, users can keep traversing “Word” objects through “Line” objects’ class object methods. Here are the class objects

Class Type	Description
File	Object to save file data.
Line	Object to save line data in a file.
Word	Object to save word data in a line

There is one type of class object – Macro. The object is used to get detailed information about a macro. You can get a “Macro” object through text_macro method in “Word” class.

Note:
For a nested macro, you can only get the first macro information due to a limitation.

Class Type

File

***class* text.File**

line_handles()

Get line handle list

Returns: Return the handle list obtained by 1-many method: line

Return type: list of text.Line.

Example

```
>>> hdlList = ref.line_handles()
>>> hdlList[0].handle_type()
'npiTextLine'
```

line_by_number(line_no)

Get line handle by line number

Parameters: **line_no** (int) – The line number of target line handle

Returns: Return the line handle according to specified line number

Return type: class text.Line

Example

```
>>> lineHdl = fileHdl.line_by_number(5)
>>> lineHdl.handle_type()
'npiTextLine'
```

handle_type()

Get handle type

Returns: Return the string by property: handle_type

Return type: str

Example

```
>>> hdl.handle_type()
'obj'
```

file_name()

Get file name

Returns: Return the string by property: file_name

Return type:: str

Example

```
>>> hdl.file_name()
'test.v'
```

file_full_name()

Get file full name

Returns: Return the string by property: file_full_name

Return type: str

Example

```
>>> hdl.file_full_name()
'obj'
```

file_elab_name()

Get file elab name

Returns:Return the string by property: file_elab_name

Return type: str

Example

```
>>> hdl.file_elab_name()
'test.v#$#1'
```

file_content()

Get file content

Returns: Return the string by property: file_content

Return type: str

Example

```
>>> hdl.file_content()
'obj'
```

line_count()

Get line count

Returns: Return the integer by property: line_count

Return type: int

Example

```
>>> hdl.line_count()
1
```

file_has_vcs_protect()

Get file has vcs protect

Returns: Return the integer by property: file_has_vcs_protect

Return type int

Example

```
>>> hdl.file_has_vcs_protect()
1
```

Feedback

Line

***class* text.Line**

file_handle()

Get file handle

Returns: Return the handle obtained by 1-1 method: file

Return type: class text.File

Example

```
>>> hdl = ref.file_handle()
>>> hdl.handle_type()
'npiTextFile'
```

word_handles()

Get word handle list

Returns: Return the handle list obtained by 1-many method: word

Return type list of text.Word

Example

```
>>> hdlList = ref.word_handles()
>>> hdlList[0].handle_type()
'npiTextWord'
```

prev_line()

Get the previous line handle

Returns: Return the previous line handle

Return type: class text.File

Example

```
>>> lineHdl = fileHdl.line_by_number(5)
>>> prevLineHdl = lineHdl.prev_line()
>>> prevLineHdl.line_number()
'4'
```

next_line()

Get the next line handle

Returns: Return the next line handle

Return type: class text.File

Feedback

Example

```
>>> lineHdl = fileHdl.line_by_number(5)
>>> nextLineHdl = lineHdl.next_line()
>>> nextLineHdl.line_number()
'6'
```

insert_line_before(*new_line*)

Insert new line string before the current handle

Parameters: **new_line (str)** – The new line string to insert

Returns: Return the new inserted line handle

Return type: class text.File

Example

```
>>> insertHdl = lineHdl.insert_line_before(' mm m0();')
>>> insertHdl.handle_type()
'npiTextLine'
```

insert_line_after(*new_line*)

Insert new line string after the current handle

Parameters: **new_line (str)** – The new line string to insert

Returns: Return the new inserted line handle

Return type: class text.File

Example

```
>>> insertHdl = lineHdl.insert_line_after(' mm m0();')
>>> insertHdl.handle_type()
'npiTextLine'
```

replace_line(*new_line*)

Replace the current line handle with new line string

Parameters: **new_line (str)** – The new line string to replace

Returns: Return the new replaced line handle

Return type: class text.File

Example

```
>>> replacedHdl = lineHdl.replace_line(' mm m0();')
>>> replacedHdl.line_content()
' mm m0();'
```

Feedback

handle_type()

Get handle type

Returns: Return the string by property: handle_type

Return type: str

Example

```
>>> hdl.handle_type()
'obj'
```

line_number()

Get line number

Returns: Return the integer by property: line_number

Return type: int

Example

```
>>> hdl.line_number()
1
```

line_content()

Get line content

Returns: Return the string by property: line_content

Return type: str

Example

```
>>> hdl.line_content()
' mm m0(a, b, c);'
```

word_count()

Get word count

Returns: Return the integer by property: word_count

Return type: int

Example

```
>>> hdl.word_count()
1
```

Feedback

Word

***class* text.Word**

line_handle()

Get line handle

Returns Return the handle obtained by 1-1 method: line

Return type: class text.Line

Example

```
>>> hdl = ref.line_handle()
>>> hdl.handle_type()
'npiTextLine'
```

file_by_include_word()

Get file handle by include word handle

Returns: Return the file handle

Return type: class text.File

Example

```
>>> fileHdl = incWordHdl.file_by_include_word()
>>> fileHdl.handle_type()
'npiTextFile'
```

insert_word_before(new_word)

Insert word handle before the current handle

Parameters: **new_word (str)** – The new word string to insert

Returns: Return the new inserted word handle

Return type: word.Line

Example

```
>>> insertHdl = wordHdl.insert_word_before('nn')
>>> insertHdl.word_name()
'nn'
```

insert_word_after(new_word)

Insert word handle before the current handle

Parameters: **new_word (str)** – The new word string to insert

Returns: Return the new inserted word handle

Return type: word.Line

Example

```
>>> insertHdl = wordHdl.insert_word_before('nn')
>>> insertHdl.word_name()
'nn'
```

replace_word(new_word)

Replace the current word handle with new line handle

Parameters: **new_word (str)** – The new word string to replace

Returns: Return the new inserted word handle

Return type: word.Line

Example

```
>>> replacedHdl = wordHdl.replace_word('nn')
>>> replacedHdl.word_name()
'nn'
```

expand_include()

Expand the include file content

Returns: Return 1 if success; Otherwise, return 0

Return type: int

Example

```
>>> incWordHdl.expand_include()
1
```

expand_macro()

Expand the macro define content

Returns: Return 1 if success; Otherwise, return 0

Return type: int

Example

```
>>> marcoWordHdl.expand_macro()
1
```

text_macro()

Get a Macro object

Returns: Return the Macro object

Return type: text.Macro

Example

```
>>> textMacro = wordHdl.text_macro()
```

handle_type()

Get handle type

Returns:: Return the string by property: handle_type

Return type: str

Example

```
>>> hdl.handle_type()
'obj'
```

word_number()

Get word number

Returns: Return the integer by property: word_number

Return type: int

Example

```
>>> hdl.word_number()
1
```

word_name()

Get word name

Returns: Return the string by property: word_name

Return type: str

Example

```
>>> hdl.word_name()
'obj'
```

word_attribute(*is_get_enum=False*)

Get word attribute

Parameters: **is_get_enum (bool)** – Flag for the data type of return value

Returns: Return the string property if is_get_enum is False; Otherwise, return the integer property if is_get_enum is True

Return type: str or int

Example

```
>>> hdl.word_attribute()
'npiTextWhiteSpace'
>>> hdl.word_attribute(True)
2
```

Macro

***class* text.Macro**

ok()

Check whether the attribute type of the word object is npiTextMacroName

Returns: Return 1 if the attribute type of the word object is npiTextMacroName; Otherwise, return 0

Return type: int

Example

```
>>> wordHdl.word_attribute()
'npiTextMacroName'
>>> textMacro = wordHdl.text_macro()
>>> textMacro.ok()
1
```

arg_num()

Get the number of arguments of the macro

Returns: Return the number of arguments

Return type: int

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.arg_num()
2
```

arg(arg_index)

Get the argument name at a specified index of the macro

Parameters: **arg_index (int)** – Specify the index of the argument

Returns: Return the argument name at the specified index

Return type: str

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.arg(0)
`TOP(2,3)
```

value()

Get the expanded value of the macro

Returns: Return the expanded value

Return type: str

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.value()
2 + 3 - 1 + 22 + 33 - 1 - 1
```

def_file()

Get the name of the file in which the macro is defined

Returns: Return the file name

Return type: str

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.def_file()
inc.v
```

def_line()

Get the line number where the macro is defined

Returns: Return the line number

Return type: int

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.def_line()
1
```

def_arg(arg_index)

Get the argument name at a specified index in the macro definition

Parameters: **arg_index (int)** – Specify the index of the argument

Returns: Return the argument name at the specified index

Return type: str

Feedback

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.def_arg(0)
a
```

def_value()

Get the value of the macro definition

Returns: Return the value

Return type: str

Example

```
>>> textMacro = wordHdl.text_macro()
>>> textMacro.def_value()
a + b - 1
```

L0 APIs

Following are the L0 APIs in text module:

Class Type	Description
text.get_file_list()	Get file handle list.
text.file_by_name(name)	Get file handle by name.
text.file_by_module_name(name)	Get file handle by module name.
text.delete_line(line_hdl)	Delete line handle.
text.delete_word(word_hdl)	Delete word handle.

Public Functions

text.get_file_list()

Get file handle list

Returns: Return the list of file handles

Return type: list of text.File

Example

```
>>> fileList = text.get_file_list()
>>> print(fileList[0].file_name())
'top.v'
```

text.file_by_name(*name*)

Get file handle by name

Parameters: **name (str)** – the file name to get

Returns: Return the file handle according to the file name

Return type: class text.File

Example

Example

```
>>> fileHdl = text.file_by_name('top.v')
>>> fileHdl.file_name()
'top.v'
```

text.file_by_module_name(*name*)

Get file handle by module name

Parameters: **name (str)** – the module name to find the file

Returns: Return the file handle according to the module name

Return type: class text.File

Example

```
>>> fileHdl = text.file_by_module_name('top')
>>> fileHdl.file_name()
'top.v'
```

text.delete_line(*line_hdl*)

Delete line handle

Parameters: **line_hdl** (class text.Line) – the line handle to be deleted

Returns: Return 1 if successful, otherwise return 0.

Return type: int

Example

```
>>> text.delete_line(lineHdl)
1
```

Feedback

text.delete_word(*word_hdl*)

Delete word handle

Parameters: **word_hdl** (class text.Word) – the word handle to be deleted

Returns: Return 1 if successful, otherwise return 0.

Return type: int

Example

```
>>> text.delete_word(wordHdl)
1
```