vhdl-style-guide Documentation

Release 3.3.3

Jeremiah C Leary

Dec 18, 2021

Contents:

CHAPTER 1

Overview

VHDL Style Guide (VSG) provides coding style guide enforcement for VHDL code.

1.1 Why VSG?

VSG was created after participating in a code review in which a real issue was masked by a coding style issue. A finding was created for the style issue, while the real issue was missed. When the code was re-reviewed, the real issue was discovered. The coding style issue seemed to blind me to the real issue.

Depending on your process, style issues can take a lot of time to resolve.

1. Create finding/ticket/issue
2. Disposition finding/ticket/issue
3. Fix the problem
4. Verify the problem was fixed

Spending less time on style issues leaves more time to analyze code structure. Eliminating style issues reduces the amount of time performing code reviews. This results in a higher quality code base.

1.2 Key Benefits

- Explicitly define VHDL coding standards
- Make coding standards visible to everyone
- Improve code reviews
- Quickly bring code up to current standards

VSG allows the style of the code to be defined and enforced over portions or the entire code base.

1.3 Key Features

- Command line tool
  - Integrates into continuous integration flow tools
- Reports and fixes issues found
  - Horizontal whitespace
  - Vertical whitespace
  - Upper and lower case
  - Keyword alignments
  - etc...
- Fully configurable rules via JSON/YAML configuration file
  - Disable rules
  - Alter behavior of existing rules
  - Change phase of execution
- Localize rule sets
  - Create your own rules using python
  - Use existing rules as a template
  - Fully integrates into base rule set

1.4 Known Limitations

VSG is a continual work in progress. As such, this version has the following known limitations:

- Parser will not process configurations
- Parser will not process embedded PSL
- Parser will not process VHDL 2019

CHAPTER 2

Gallery

The examples shown below illustrate the formatting enforced by VSG. They show a subset of the rules:

- capitalization
- indentation
- column alignments
  - comments
  - :'s
  - assignment operators (<= and =>)
- vertical spacing

2.1 Entities

```
entity GRP_DEBOUNCER is
  generic (
    N       : positive := 8;                          -- input bus width
    CNT_VAL : positive := 10000                       -- clock counts for debounce period
  );
  port (
    CLK_I  : in    std_logic := 'X';                  -- system clock
    DATA_I : in    std_logic_vector(1 downto 0)       -- noisy input data
    DATA_O : out   std_logic_vector(1 downto 0);      -- registered stable output data
    STRB_O : out   std_logic                          -- strobe for new data available
  );
end entity GRP_DEBOUNCER;
```

2.2 Architectures

```
architecture BEHAVIORAL of PIC is

  type state_type is (
    reset_s, get_commands, jump_int_method, start_polling,
    ack_txinfo_rxd, start_priority_check, tx_int_info_priority
  );

  signal next_s               : state_type :=reset_s;
  signal int_type             : unsigned(1 downto 0):="01";
  signal int_index, count_cmd : integer := 0;

  type prior_table is array (0 to 7) of unsigned(2 downto 0);

  signal pt                   : prior_table := (others => (others => '0'));
  signal int_pt               : unsigned(2 downto 0):="000";
  signal flag,      flag1     : std_logic := '0';

begin

end architecture BEHAVIORAL;
```

2.3 Component Declarations

```
component CPU is
  port (
    CLK_I         : in    std_logic;
    SWITCH        : in    std_logic_vector(9 downto 0);

    SER_IN        : in    std_logic;
    SER_OUT       : out   std_logic;

    TEMP_SPO      : in    std_logic;
    TEMP_SPI      : out   std_logic;
    TEMP_CE       : out   std_logic;
    TEMP_SCLK     : out   std_logic;

    SEG1          : out   std_logic_vector(7 downto 0);
    SEG2          : out   std_logic_vector( 7 downto 0);
    LED           : out   std_logic_vector( 7 downto 0);

    XM_ADR        : out   std_logic_vector(15 downto 0);
    XM_RDAT       : in    std_logic_vector( 7 downto 0);
    XM_WDAT       : out   std_logic_vector( 7 downto 0);
    XM_WE         : out   std_logic;
    XM_CE         : out   std_logic
  );
end component;
```

2.4 Component Instantiations

```
INTERLEAVER_I0 : INTERLEAVER
  generic map (
    DELAY        => TREL1_LEN + TREL2_LEN + 2 + delay,
    WAY          => 0
  )
  port map (
    CLK          => clk,
    RST          => rst,
    D            => tmp0,
    Q            => tmp1
  );
```

2.5 Concurrent Assignments

```
nCounter        <= x"FFFFFF" when Counter=x"FFFFFF" and Button='1' else
                   x"000000" when Counter=x"000000" and Button='0' else
                   Counter + 1 when Button='1' else
                   Counter - 1;
nextHistory     <= '0' when Counter=x"000000" else
                   '1';
nButtonHistory <= nextHistory & ButtonHistory(1);
Dout            <= '1' when ButtonHistory="01" else
                   '0';
```

CHAPTER 3

Installation

There are two methods to install VSG.

3.1 PIP

The most recent released version is hosted on PyPI. It can be installed using **pip**.

```
pip install vsg
```

This is the preferred method for installing VSG.

3.2 Git Hub

The latest development version can be cloned from the git hub repo.

```
git clone https://github.com/jeremiah-c-leary/vhdl-style-guide.git
```

Then installed using the setup.py file.

```
python setup.py install
```

CHAPTER 4

Usage

VSG is a both a command line tool and a python package. The command line tool can be invoked with:

```
$ vsg
usage: VHDL Style Guide (VSG) [-h] [-f FILENAME [FILENAME ...]] [-lr LOCAL_RULES] [-c␣
↪CONFIGURATION [CONFIGURATION ...]] [--fix]
                              [-fp FIX_PHASE] [-j JUNIT] [-js JSON] [-of {vsg,
↪syntastic,summary}] [-b] [-oc OUTPUT_CONFIGURATION]
                              [-rc RULE_CONFIGURATION] [--style {indent_only,jcl}] [-
↪v] [-ap] [--fix_only FIX_ONLY] [-p JOBS]
                              [--debug]

Analyzes VHDL files for style guide violations. Reference documentation is located␣
↪at: http://vhdl-style-guide.readthedocs.io/en/latest/index.html

optional arguments:
  -h, --help            show this help message and exit
  -f FILENAME [FILENAME ...], --filename FILENAME [FILENAME ...]
                        File to analyze
  -lr LOCAL_RULES, --local_rules LOCAL_RULES
                        Path to local rules
  -c CONFIGURATION [CONFIGURATION ...], --configuration CONFIGURATION [CONFIGURATION .
↪..]
                        JSON or YAML configuration file(s)
  --fix                 Fix issues found
  -fp FIX_PHASE, --fix_phase FIX_PHASE
                        Fix issues up to and including this phase
  -j JUNIT, --junit JUNIT
                        Extract Junit file
  -js JSON, --json JSON
                        Extract JSON file
  -of {vsg,syntastic,summary}, --output_format {vsg,syntastic,summary}
                        Sets the output format.
  -b, --backup          Creates a copy of input file for comparison with fixed␣
↪version.
```

(continues on next page)

(continued from previous page)

```
  -oc OUTPUT_CONFIGURATION, --output_configuration OUTPUT_CONFIGURATION
                        Write configuration to file name.
  -rc RULE_CONFIGURATION, --rule_configuration RULE_CONFIGURATION
                        Display configuration of a rule
  --style {indent_only,jcl}
                        Use predefined style
  -v, --version         Displays version information
  -ap, --all_phases     Do not stop when a violation is detected.
  --fix_only FIX_ONLY   Restrict fixing via JSON file.
  -p JOBS, --jobs JOBS  number of parallel jobs to use, default is the number of cpu␣
↪cores
  --debug               Displays verbose debug information
```

Command Line Options

Option	Description
-f FILENAME	The VHDL file to be analyzed or fixed. Multiple files can be passed through this option.
–local_rules LOCAL_RULES	Additional rules not in the base set.
–configuration CONFIGURATION	JSON or YAML file(s) which alters the behavior of VSG. Configuration can also include a list files to analyze. Any combination of JSON and YAML files can be passed. Each will be processed in order from left to right.
–fix	Update issues found. Replaces current file with updated one.
–fix_phase	Applies for all phases up to and including this phase. Analysis will then be performed on all phases.
–junit	Filename of JUnit XML file to generate.
–json	Filename of JSON file to generate.
–output_format	**Configures the sdout output format.** vsg – standard VSG output syntastic – format compatible with the syntastic VIM module summary – Minimal output useful when running on multiple files
–backup	Creates a copy of the input file before applying any fixes. This can be used to compare the fixed file against the original.
–output_configuration	Writes a JSON configuration file of the current run. It includes a file_list, local_rules (if used), and how every rule was configured. This configuration can be fed back into VSG.
–rule_configuration	Displays the configuration of a rule.
–style	Use a built in coding style.
–version	Displays the version of VSG.
–all-phases	Executes all phases without stopping if a violation is found.
–fix_only	Restrict which rules are fixed based on JSON file.
–jobs	Restrict the number of cores used to run. The default is the number of cores available.
–debug	Print verbose debug information to assist with debuging errors with VSG.

Here is an example output running against a test file:

```
$ vsg -f example/architecture-empty.vhd
================================================================================
File:  example/architecture-empty.vhd
================================================================================
Phase 1 of 7... Reporting
Total Rules Checked: 83
Total Violations:     3
  Error   :      3
  Warning :      0
----------------------------+------------+------------+------------------------------
↪-------
  Rule                      |  severity  |  line(s)   | Solution
----------------------------+------------+------------+------------------------------
↪-------
  port_021                  | Error      |         45 | Move the ( to the same line␣
↪as the "port" keyword.
  instantiation_034         | Error      |        169 | Change to component␣
↪instantiation
  generic_map_003           | Error      |        170 | Move the ( to the same line␣
↪as the "generic map" keyword.
----------------------------+------------+------------------------------------------
NOTE: Refer to online documentation at https://vhdl-style-guide.readthedocs.io/en/
↪latest/index.html for more information.
```

VSG will report the rule which is violated and the line number or group of lines where the violation occured. It also gives a suggestion on how to fix the violation. The rules VSG uses are grouped together into *Phases*. These phases follow the order in which the user would take to address the violations. Each rule is detailed in the *Rules* section. The violation and the appropriate fix for each rule is shown.

The violations can be fixed manually, or use the **–fix** option to have VSG update the file.

```
$ vsg -f example/architecture-empty.vhd
================================================================================
File:  example/architecture-empty.vhd
================================================================================
Phase 7 of 7... Reporting
Total Rules Checked: 378
Total Violations:     0
```

If rule violations can not be fixed, they will be reported after fixing everything else:

```
$ vsg -f example/architecture-empty.vhd
================================================================================
File:  example/architecture-empty.vhd
================================================================================
Phase 1 of 7... Reporting
Total Rules Checked: 83
Total Violations:     1
  Error   :      1
  Warning :      0
----------------------------+------------+------------+------------------------------
↪-------
  Rule                      |  severity  |  line(s)   | Solution
----------------------------+------------+------------+------------------------------
↪-------
  instantiation_034         | Error      |        169 | Change to component␣
↪instantiation
```

(continues on next page)

(continued from previous page)

```
----------------------------+------------+------------------------------------
NOTE: Refer to online documentation at https://vhdl-style-guide.readthedocs.io/en/
↪latest/index.html for more information.
```

4.1 Error Codes

One of the following error codes will be returned after running VSG:

Error Code	Description
0	VSG ran without encountering any errors and no rule violations were detected.
1	VSG ran and detected a rule violation.
2	An attempt was made to configure a rule which was depricated.

CHAPTER 5

Formatting Terminal Output

VSG supports multiple display output using the *-of* command line argument.

Option	Description
vsg	Default output format.
syntastic	Output format following the syntastic standard. Useful for integrating with Vim.
summary	Output format showing the results at the file level.

5.1 VSG

This is the default output format of VSG. It gives analysis statistics along with individual rule violations. This format is the most verbose of all output formats.

Here is a sample output:

```
================================================================================
File:  design_fixed/BufFifo/BUF_FIFO.vhd
================================================================================
Phase 1 of 7... Reporting
Total Rules Checked: 83
Total Violations:    17
  Error   :    17
  Warning :     0
----------------------------+------------+------------+--------------------------------
↪-------
  Rule                      |  severity  |  line(s)   | Solution
----------------------------+------------+------------+--------------------------------
↪-------
  port_021                  | Error      |         45 | Move the ( to the same line␣
↪as the "port" keyword.
  instantiation_034         | Error      |        169 | Change to component␣
↪instantiation
```

(continues on next page)

(continued from previous page)

```
  generic_map_003            | Error      |          170 | Move the ( to the same line␣
↪as the "generic map" keyword.
  port_map_003               | Error      |          175 | Move the ( to the same line␣
↪as the "port map" keyword.
  instantiation_034          | Error      |          186 | Change to component␣
↪instantiation
  instantiation_034          | Error      |          196 | Change to component␣
↪instantiation
  generic_map_003            | Error      |          197 | Move the ( to the same line␣
↪as the "generic map" keyword.
  port_map_003               | Error      |          202 | Move the ( to the same line␣
↪as the "port map" keyword.
  instantiation_034          | Error      |          213 | Change to component␣
↪instantiation
  generic_map_003            | Error      |          214 | Move the ( to the same line␣
↪as the "generic map" keyword.
  port_map_003               | Error      |          219 | Move the ( to the same line␣
↪as the "port map" keyword.
  process_012                | Error      |          231 | Add *is* keyword
  if_002                     | Error      |          313 | Enclose condition in ()'s.
  process_012                | Error      |          337 | Add *is* keyword
  if_002                     | Error      |          366 | Enclose condition in ()'s.
  process_012                | Error      |          376 | Add *is* keyword
  if_002                     | Error      |          455 | Enclose condition in ()'s.
----------------------------+------------+------------+------------------------------
↪-------
NOTE: Refer to online documentation at https://vhdl-style-guide.readthedocs.io/en/
↪latest/index.html for more information.
```

5.2 Synastic

Using the syntastic format allows editors with understand that standard to use the output of VSG.

Below is the output format definition:

```
<status>: <filename>(<line_number>)<rule> -- <solution>
```

Where:

Item	Description
status	ERROR = Violation. WARNING = Non Violation.
filename	The file being analyzed.
line_number	The line number the violation occured.
rule	The rule id that detected the violation
solution	A description of how to fix the violation

Here is a sample output using the **syntastic** option:

```
ERROR: design_fixed/mdct/DBUFCTL.VHD(38)entity_017 -- Move : -1 columns
ERROR: design_fixed/mdct/DBUFCTL.VHD(59)process_035 -- Move 13 columns
ERROR: design_fixed/mdct/DCT2D.VHD(329)instantiation_033 -- Add *component* keyword
```

(continues on next page)

(continued from previous page)

```
ERROR: design_fixed/mdct/MDCT.VHD(83)instantiation_034 -- Change to component␣
↪instantiation
ERROR: design_fixed/mdct/RAM.VHD(36)entity_017 -- Move : -12 columns
```

5.3 Summary

Using the summary format will display results at the file level. Individual rule violations will not be displayed.

Below is the output format definition:

```
File: <filename> <status> (<num_rules> rules checked) [<severity>: <num_severity>] ...
```

Where:

Item	Description
filename	The file being analyzed.
status	OK = No violations detected. ERROR = Violations detected.
num_rules	The number of rules checked before a violation was detected.
severity	The severity type being reported.
num_severity	The number of violations of that severity type

Note: The <severity> and <num_severity> will be repeated for each severity type.

Here is a sample output using the **summary** option:

```
File: design/top/JpegEnc.vhd ERROR (83 rules checked) [Error: 23] [Warning: 0]
File: design/BufFifo/SUB_RAMZ.VHD OK (329 rules checked) [Error: 0] [Warning: 0]
File: design/common/RAMZ.VHD OK (329 rules checked) [Error: 0] [Warning: 0]
File: design/mdct/DBUFCTL.VHD OK (329 rules checked) [Error: 0] [Warning: 0]
File: design/mdct/DCT2D.VHD ERROR (83 rules checked) [Error: 1] [Warning: 0]
```

Any line with an ERROR will be reported to stderr. Any line with an OK will be reported to stdout.

CHAPTER 6

Styles

VSG supports several predefined styles. They can be used with the **–style** command line option.

The table below lists the built in styles available

Style	Description
indent_only	Only applies indent rules
jcl	Coding style preferred by Jeremiah Leary

6.1 Style Descriptions

6.1.1 indent_only

This style only applies indenting rules.

This style attempts to improve readability by:

- Indenting
  - 2 spaces

6.1.2 jcl

This style was in affect before the 2.0.0 release. It maintains the same style as new rules are added.

This style attempts to improve readability by:

- Emphasising non vhdl identifiers by capitalizing them.
  - entity names
  - architecture names
  - ports

    - generics
    - etc. . .
- Blank lines added between major items
    - processes
    - if statements
    - case statements
- Alignments
    - :'s over entire entities, components, instantiations, etc. . .
    - <='s over groups of sequential statements
    - inline comments within processes, architecture declarative regions, etc. . .
- Indenting
    - 2 spaces
- Structure
    - No single line sequential statements using the when keyword
    - No code after the case when statements
    - Split if/elsif/else/end if into separate lines
    - Removing comments from instantiation and component ports and generics
    - No more than two signals can be declared on a single line

6.2 Adjusting built in styles

The built in styles provide several examples of how VHDL code can be formatted to improve readability. This is by no means the only way. The styles can be modified using the **–configuration** option.

Follow these steps to adjust the styles to the local flavor:

1. Pick a style that is close to yours
2. Create a configuration to modify the rules which must change
3. Use the style and configuration to analyze your code

6.2.1 Example

Let us assume the jcl style matches 95% of the desired style. The only differences are:

- The entity keyword is always lower case
- Indenting is three spaces instead of two

Create a configuration with the following:

```
---
rule:
  global:
    indentSize: 3
```

(continues on next page)

(continued from previous page)

```
  entity_004:
    case: lower
...
```

Then use the style and configuration together:

```
$ vsg --style jcl --configuration my_config.yaml -f fifo.vhd
```

CHAPTER 7

Configuring

VSG can use a configuration file to alter it's behavior and/or include a list of files to analyze. This is accomplished by passing JSON and/or YAML file(s) through the **–configuration** command line argument. This is the basic form of a configuration file in JSON:

```
{
    "file_list":[
      "fifo.vhd",
      "$PATH_TO_FILE/spi_master.vhd",
      "$OTHER_PATH/src/*.vhd",
      "source/spi.vhd": {
        "rule": {
          "ruleId_ruleNumber":"blah"
      }
    ],
    "local_rules":"$DIRECTORY_PATH",
    "rule":{
        "global":{
            "attributeName":"AttributeValue"
        },
        "ruleId_ruleNumber":{
            "attributeName":"AttributeValue"
        }
    }
}
```

This is the basic form of a configuration file in YAML:

```
---
file_list:
  - fifo.vhd
  - source/spi.vhd:
      rule:
        ruleId_ruleNumber:
          attributeName: AttributeValue
```

(continues on next page)

(continued from previous page)

```
  - $PATH_TO_FILE/spi_master.vhd
  - $OTHER_PATH/src/*.vhd
local_rules: $DIRECTORY_PATH
rule:
  global:
    attributeName: AttributeValue
  ruleId_ruleNumber:
    attributeName: AttributeValue
...
```

It is not required to have **file_list**, **local_rules**, and **rule** defined in the configuration file. Any combination can be defined, and the order does not matter.

Note: All examples of configurations in this documentation use JSON. However, YAML can be used instead.

7.1 file_list

The file_list is a list of files that will be analyzed. Environment variables will expanded. File globbing is also supported. The Environment variables will be expanded before globbing occurs. This option can be useful when running VSG over multiple files.

Rule configurations can be specified for each file by following the format of the **rule** configuration.

7.2 local_rules

Local rules can be defined on the command line or in a configuration file. If they are defined in both locations, the configuration will take precedence.

7.3 rule

Any attribute of any rule can be configured. Using **global** will set the attribute for every rule. Each rule is addressable by using it's unique **ruleId** and **ruleNumber** combination. For example, whitespace_006 or port_010.

Note: If **global** and unique attributes are set at the same time, the unique attribute will take precedence.

Here are a list of attributes that can be altered for each rule:

Attribute	Values	Description
indentSize	Integer	Sets the number of spaces for each indent level.
phase	Integer	Sets the phase the rule will run in.
disable	Boolean	If set to True, the rule will not run.
fixable	Boolean	If set to False, the violation will not be fixed

7.4 Reporting Single Rule Configuration

The configuration for a single rule can be reported using the **-rc** option:

```
$ vsg -rc entity_001
{
  "rule": {
    "entity_001": {
      "indentSize": 2,
      "phase": 4,
      "disable": false,
      "fixable": true
    }
  }
}
```

VSG will print the configuration for the rule given in a JSON format. This configuration can be altered and added to a configuration file.

7.5 Reporting Configuration for All Rules

Every rule configuration can be report and saved to a file using the **-oc** option:

```
$ vsg -oc configuration.json
```

The output file will be in JSON format and can be modified and passed back to VSG using the *-c* option.

7.6 Rule Configuration Priorities

There are three ways to configure a rule. From least to highest priority are:

- **[rule][global]**
- **[rule][<identifier>]**
- **[file_list][<filename>][rule][<identifier>]**.

If the same rule is defined in all three locations as in the example below, then the final setting will be equal to the highest priority.

```
{
  "file_list":[
    "entity.vhd":{
      "rule":{
        "length_001":{
          "disable": true
        }
      }
    },
    "architecture.vhd",
    "package.vhd"
  ],
  "rule":{
    "global":{
```

(continues on next page)

(continued from previous page)

```
      "disable": true
    },
    "rule": {
      "length_001":{
        "disable": false
    }
  }
}
```

In this example configuration, all rules are disabled by the **global** configuration. Then rule **length_001** is enabled for the files **architecture.vhd**, **package.vhd** and **entity.vhd** by the **rule** configuration. Then rule **length_001** is disabled for the file **entity.vhd**.

7.7 Example: Disabling a rule

Below is an example of a JSON file which disables the rule **entity_004**

```
{
    "rule":{
        "entity_004":{
            "disable":true
        }
    }
}
```

Use the configuration with the **–configuration** command line argument:

```
$ vsg -f RAM.vhd --configuration entity_004_disable.json
```

7.8 Example: Setting the indent increment size for a single rule

The indent increment size is the number of spaces an indent level takes. It can be configured on an per rule basis. . .

```
{
    "rule":{
        "entity_004":{
            "indentSize":4
        }
    }
}
```

7.9 Example: Setting the indent increment size for all rules

Configure the indent size for all rules by setting the **global** attribute.

```
{
    "rule":{
        "global":{
            "indentSize":4
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

7.10 Multiple configurations

More than one configuration can be passed using the **–configuration** option. This can be useful in two situations:

1) Block level configurations
2) Multilevel rule configurations

The priority of the configurations is from right to left. The last configuration has the highest priority. This is true for all configuration parameters except **file_list**.

7.10.1 Block level configurations

Many code bases are large enough to be broken into multiple sub blocks. A single configuration can be created and maintained for each subblock. This allows each subblock to be analyzed independently.

When the entire code base needs be analyzed, all the subblock configurations can be passed to VSG. This reduces the amount of external scripting required.

config_1.json

```
{
    "file_list":[
      "fifo.vhd",
      "source/spi.vhd",
      "$PATH_TO_FILE/spi_master.vhd",
      "$OTHER_PATH/src/*.vhd"
    ]
}
```

config_2.json

```
{
    "file_list":[
      "dual_port_fifo.vhd",
      "flash_interface.vhd",
      "$PATH_TO_FILE/ddr.vhd"
    ]
}
```

Both configuration files can be processed by vsg with the following command:

```
$ vsg --configuration config_1.json config_2.json
```

7.10.2 Multilevel rule configurations

Some code bases may require rule adjustments that apply to all the files along with rule adjustments against individual files. Use multiple configurations to accomplish this. One configuration can handle code base wide adjustments. A second configuration can target individual files. VSG will combine any number of configurations to provide a unique set of rules for any file.

config_1.json

```
{
    "rule":{
        "entity_004":{
            "disable":true
        },
        "entity_005":{
            "disable":true
        },
        "global":{
            "indentSize":2
        }
    }
}
```

config_2.json

```
{
    "rule":{
        "entity_004":{
            "disable":false,
            "indentSize":4
        }
    }
}
```

Both configuration files can be processed by VSG with the following command:

```
$ vsg --configuration config_1.json config_2.json -f fifo.vhd
```

VSG will combine the two configurations into this equivalent configuration…

```
{
    "rule":{
        "entity_004":{
            "disable":false,
            "indentSize":4
        },
        "entity_005":{
            "disable":true
        },
        "global":{
            "indentSize":2
        }
    }
}
```

…and run on the file **fifo.vhd**.

7.11 Configuring Disabled Rules

Each rule is either enabled (actively checked) or disabled (not checked). Each rule can be enabled or disabled by user configuration.

Most rules are enabled by default while some are disabled by default. Rules disabled by default are marked by and are typically naming convention rules. They can be enabled by setting the *disable* option to *False* in a configuration.

```
rule :
  <rule_id>:
      disable: False
```

7.11.1 Rules Disabled by Default

- after_001
- after_002
- after_003
- architecture_025
- block_600
- block_601
- block_comment_001
- block_comment_002
- block_comment_003
- comment_011
- constant_015
- generate_017
- generic_020
- instantiation_600
- instantiation_601
- package_016
- package_017
- package_body_600
- package_body_601
- port_011
- port_025
- process_036
- signal_008
- subtype_004
- type_015
- variable_012

7.12 Configuring Uppercase and Lowercase Rules

There are several rules that enforce either uppercase or lowercase. The default for all such rules is `lowercase`. The decision was motivated by the fact, that the VHDL language is case insensitive. Having the same default for all case rules also results in less documentation and code to maintain. The default value for each of these case rules can be overridden using a configuration.

7.12.1 Overriding Default Lowercase Enforcement

The default lowercase setting can be changed using a configuration.

For example the rule constant_002 can be changed to enforce uppercase using the following configuration:

```
---

rule :
  constant_002 :
     case : 'upper'
```

7.12.2 Changing Multiple Case Rules

If there are a lot of case rules you want to change, you can use the global option to reduce the size of the configuration. For example, if you want to uppercase everything except the entity name, you could write the following configuration:

```
---

rule :
  global :
    case : 'upper'
  entity_008 :
    case : 'lower'
```

7.12.3 Rules Enforcing Case

- architecture_004
- architecture_009
- architecture_011
- architecture_013
- architecture_014
- architecture_019
- architecture_020
- architecture_021
- architecture_028
- attribute_declaration_500
- attribute_declaration_501
- attribute_declaration_502

- attribute_specification_500
- attribute_specification_501
- attribute_specification_502
- attribute_specification_503
- block_500
- block_501
- block_502
- block_503
- block_504
- block_505
- block_506
- case_014
- case_015
- case_016
- case_017
- case_018
- component_004
- component_006
- component_008
- component_010
- component_012
- component_014
- constant_002
- constant_004
- constant_011
- constant_013
- context_004
- context_012
- context_013
- context_014
- context_015
- context_016
- context_ref_003
- context_ref_004
- entity_004
- entity_006

- entity_008
- entity_010
- entity_012
- entity_014
- entity_specification_500
- entity_specification_501
- entity_specification_502
- entity_specification_503
- file_statement_002
- for_loop_003
- function_004
- function_005
- function_010
- function_013
- function_014
- function_017
- generate_005
- generate_009
- generate_010
- generate_012
- generic_007
- generic_009
- generic_017
- generic_map_001
- generic_map_002
- if_statement_025
- if_statement_026
- if_statement_027
- if_statement_028
- if_statement_029
- if_statement_034
- instantiation_008
- instantiation_009
- instantiation_027
- instantiation_031
- library_004

- library_005
- package_004
- package_006
- package_008
- package_010
- package_013
- package_018
- package_body_500
- package_body_501
- package_body_502
- package_body_503
- package_body_504
- package_body_505
- package_body_506
- package_body_507
- port_010
- port_017
- port_018
- port_019
- port_map_001
- port_map_002
- procedure_007
- procedure_008
- procedure_009
- process_004
- process_005
- process_008
- process_009
- process_013
- process_017
- process_019
- range_001
- range_002
- signal_002
- signal_004
- signal_010

- signal_011
- signal_014
- subtype_002
- type_definition_002
- type_definition_004
- type_definition_013
- type_definition_014
- variable_002
- variable_004
- variable_010
- variable_011

7.13 Configuring Prefix and Suffix Rules

There are several rules that enforce specific prefixes or suffixes in different name identifiers. It is noted in the documentation, what the default prefixes and suffixes are for each such rule.

All prefix and suffix rules are disabled by default. The defaults for each of these rules can be overridden using a configuration.

Note: Some elements have both prefix and suffix rules. Depending on the desired style, either or both can be enabled.

7.13.1 Overriding Default Prefix Enforcement

The default setting can be changed using a configuration. The rule variable_012 defaults to following prefix: ['v_']. We can use the following configuration to change allowed prefix:

```
---

 rule :
     variable_012:
         # Each prefix rule needs to be enabled explicitly.
         disable: false
         prefixes: ['var_']
```

7.13.2 Overriding Default Suffix Enforcement

The default setting can be changed using a configuration. For example, the rule port_025 defaults to following suffixes: ['_I', '_O', '_IO']. We can use the following configuration to change allowed suffixes:

```
---

 rule :
     port_025:
         # Each suffix rule needs to be enabled explicitly.
```

(continues on next page)

(continued from previous page)

```
        disable: false
        suffixes: ['_i', '_o']
```

7.13.3 Rules Enforcing Prefixes and Suffixes

Element	Prefix Rule	Suffix Rule
Block Label	block_601	block_600
Constant Identifier	constant_015	constant_600
Generate Label	generate_017	generate_600
Generic Identifier	generic_020	generic_600
Package Identifier	package_017	package_016
Package Body Identifier	package_body_601	package_body_600
Port Identifier	port_011	port_025
Process Label	process_036	process_600
Signal Identifier	signal_008	signal_600
Subtype Identifier	subtype_004	subtype_600
Type Identifier	type_definition_015	type_definition_600
Variable Identifier	variable_012	variable_600

7.14 Configuring Number of Signals in Signal Declaration

VHDL allows of any number of signals to be declared within a single signal declaration. While this may be allowed, in practice there are limits impossed by the designers. Limiting the number of signals declared improves the readability of VHDL code.

The default number of signals allowed, 2, can be set by configuring rule **signal_015**.

7.14.1 Overriding Number of Signals

The default setting can be changed using a configuration. We can use the following configuration to change the number of signals allowed to 1.

```
---

rule :
  signal_015 :
     consecutive : 1
```

7.14.2 Rules Enforcing Number of Signals

- signal_015

7.15 Configuring Length Rules

VSG includes several rules enforcing maximum lengths of code structures. These rules are set as warnings.

7.15.1 Overriding Line Length

Limiting the line length of the VHDL code can improve readability. Code that exceeds the editor window is more difficult to read. The default line length is 120, and can be changed by configuring rule **length_001**.

Use the following configuration to change the line length to 180.

```
rule :
  length_001 :
     length : 180
```

7.15.2 Overridding File Line Length

Limiting the length of a VHDL file can improve readability. Excessively long files can indicate the file can be broken into smaller modules. The default line length is 2000, and can be changed by configuring rule **length_002**.

Use the following configuration to change the file length to 5000.

```
rule :
  length_002 :
     length : 5000
```

7.15.3 Overridding Process Line Length

Limiting the length of a VHDL processes can improve readability. Processes should perform a limited number of functions. Smaller processes are easier to understand.

The default length is 500 lines, and can be changed by configuring rule **length_003**.

Use the following configuration to change the process length to 1000.

```
rule :
  length_003 :
     length : 1000
```

7.15.4 Rules Enforcing Lengths

- length_001
- length_002
- length_003

7.16 Configuring Keyword Alignment Rules

There are several rules that enforce alignment for a group of lines based on the keywords such as 'after', '<=' etc. Some of the configurations are available in all keyword alignment rules, while others are rule specific.

7.16.1 Common Keyword Alignment Configuration

Following configuration options can be independently changed for each of the keyword alignment rules.

1. `compact_alignment` - if set to `True` it enforces single space before alignment keyword in the line with the longest part before the keyword. Otherwise the alignment occurs to the keyword maximum column. By default set to `True`.

   Violation

   ```
   signal sig_short    : std_logic;
   signal sig_very_long       : std_logic;
   ```

   Fix (compact_alignment = True)

   ```
   signal sig_short     : std_logic;
   signal sig_very_long : std_logic;
   ```

   Fix (compact_alignment = False)

   ```
   signal sig_short           : std_logic;
   signal sig_very_long       : std_logic;
   ```

2. `blank_line_ends_group` - if set to `True` any blank line encountered in the VHDL file ends the group of lines that should be aligned and starts new group. By default set to `True`.

   Violation

   ```
   signal wr_en : std_logic;
   signal rd_en   : std_logic;

   constant c_short_period : time;
   constant c_long_period : time;
   ```

   Fix (blank_line_ends_group = True)

   ```
   signal wr_en   : std_logic;
   signal rd_en   : std_logic;

   constant c_short_period : time;
   constant c_long_period  : time;
   ```

   Fix (blank_line_ends_group = False)

   ```
   signal wr_en            : std_logic;
   signal rd_en            : std_logic;

   constant c_short_period : time;
   constant c_long_period  : time;
   ```

3. `comment_line_ends_group` - if set to `True` any purely comment line in the VHDL file ends the group of lines that should be aligned and starts new group. By default set to `True`.

   Violation

   ```
   port (
       sclk_i : in std_logic;
       pclk_i : in std_logic;
       rst_i : in std_logic;
   ```

(continues on next page)

(continued from previous page)

```
    ---- serial interface ----
    spi_ssel_o : out std_logic;
    spi_sck_o : out std_logic;
    spi_mosi_o : out std_logic;
    spi_miso_i : in std_logic
);
```

Fix (comment_line_ends_group = True)

```
port (
    sclk_i : in std_logic;
    pclk_i : in std_logic;
    rst_i  : in std_logic;
    ---- serial interface ----
    spi_ssel_o : out std_logic;
    spi_sck_o  : out std_logic;
    spi_mosi_o : out std_logic;
    spi_miso_i : in std_logic
);
```

Fix (comment_line_ends_group = False)

```
port (
    sclk_i     : in std_logic;
    pclk_i     : in std_logic;
    rst_i      : in std_logic;
    ---- serial interface ----
    spi_ssel_o : out std_logic;
    spi_sck_o  : out std_logic;
    spi_mosi_o : out std_logic;
    spi_miso_i : in std_logic
);
```

Note: As all keyword alignment rules have above configurations they are not mentioned in the documentation for each rule.

7.16.2 Rule Specific Keyword Alignment Configuration

1. `separate_generic_port_alignment` - if set to `True` alignment within the generic declarative/mapping part is separated from alignment within the port declarative/mapping part. By default set to `True`.

   Violation

```
generic (
    g_width : positive;
    g_output_delay : positive
);
port (
    clk_i : in std_logic;
    data_i : in std_logic;
    data_o : in std_logic
);
```

   Fix (separate_generic_port_alignment = True)

```
generic (
    g_width        : positive;
    g_output_delay : positive
);
port (
    clk_i  : in std_logic;
    data_i : in std_logic;
    data_o : in std_logic
);
```

Fix (separate_generic_port_alignment = False)

```
generic (
    g_width        : positive;
    g_output_delay : positive
);
port (
    clk_i          : in std_logic;
    data_i         : in std_logic;
    data_o         : in std_logic
);
```

2. `if_control_statements_end_group` - if set to `True` any line with if control statement ends the group of lines that should be aligned and starts new group. By default set to `True`.

   Violation

```
if condition = '1' then
    data_valid <= '1';
    data <= '1';
else
    data_valid <= '0';
    hold_transmission <= '1';
end if;
```

   Fix (if_control_statements_end_group = True)

```
if condition = '1' then
    data_valid <= '1';
    data       <= '1';
else
    data_valid        <= '0';
    hold_transmission <= '1';
end if;
```

   Fix (if_control_statements_end_group = False)

```
if condition = '1' then
    data_valid        <= '1';
    data              <= '1';
else
    data_valid        <= '0';
    hold_transmission <= '1';
end if;
```

3. `case_control_statements_end_group` - if set to `True` any line with case control statement ends the group of lines that should be aligned and starts new group. By default set to `True`.

   Violation

```
case A is
    when A =>
        X <= F;
        XY <= G;
        XYZ <= H;
    when B =>
        a <= I;
        ab <= h;
        c <= a;
    when others =>
      null;
end case
```

Fix (case_control_statements_end_group = True)

```
case A is
    when A =>
        X   <= F;
        XY  <= G;
        XYZ <= H;
    when B =>
        a  <= I;
        ab <= h;
        c  <= a;
    when others =>
        null;
end case
```

Fix (case_control_statements_end_group = False)

```
case A is
    when A =>
        X   <= F;
        XY  <= G;
        XYZ <= H;
    when B =>
        a   <= I;
        ab  <= h;
        c   <= a;
    when others =>
        null;
end case
```

Note: If given keyword alignment rule has any of the above keyword alignment specific configuration, then it is explicitly noted in the documentation of this rule.

The default value for each of these case rules can be overridden using a configuration.

7.16.3 Rules Enforcing Keyword Alignment

- after_002
- architecture_026
- architecture_027

- block_401
- component_017
- component_020
- concurrent_006
- concurrent_008
- context_028
- entity_017
- entity_018
- entity_020
- function_012
- generate_401
- generate_403
- generate_405
- instantiation_010
- instantiation_029
- process_033
- process_034
- process_035
- sequential_005
- type_400
- variable_assignment_005

7.17 Configuring Identifier Alignment Rules

There are several rules that enforce alignment of identifiers in group of lines. Some of the configurations are available in all keyword alignment rules, while others are rule specific.

7.17.1 Common Identifier Alignment Configuration

Following configuration options can be independently changed for each of the identifier alignment rules.

1. `blank_line_ends_group` - if set to `True` any blank line encountered in the VHDL file ends the group of lines that should be aligned and starts new group. By default set to `True`.

   Violation

   ```
   signal wr_en   : std_logic;
   file results :

   signal rd_en   : std_logic;
   constant c_short_period : time;
   ```

   Fix (blank_line_ends_group = True)

```
signal wr_en   : std_logic;
file   results :

signal   rd_en   : std_logic;
constant c_short_period : time;
```

Fix (blank_line_ends_group = False)

```
signal   wr_en   : std_logic;
file     results :

signal   rd_en   : std_logic;
constant c_short_period : time;
```

2. `comment_line_ends_group` - if set to `True` any purely comment line in the VHDL file ends the group of lines that should be aligned and starts new group. By default set to `True`.

Violation

```
signal wr_en   : std_logic;
file results :
-- some comment
signal rd_en   : std_logic;
constant c_short_period : time;
```

Fix (comment_line_ends_group = True)

```
signal wr_en   : std_logic;
file   results :
-- some comment
signal   rd_en   : std_logic;
constant c_short_period : time;
```

Fix (comment_line_ends_group = False)

```
signal   wr_en   : std_logic;
file     results :
-- some comment
signal   rd_en   : std_logic;
constant c_short_period : time;
```

Note: As all identifier alignment rules have above configurations they are not mentioned in the documentation for each rule.

7.17.2 Rules Enforcing Identifier Alignment

- architecture_029
- block_400
- function_015
- generate-400
- generate-402

- generate-404
- package_body_400
- package_019
- procedure_010

7.18 Configuring Blank Lines

There are rules which will check for blank lines either above or below a line. These rules are designed to improve readability by separating code using blank lines.

There are a couple of options to these rules, which can be selected by using the `style` option:

Style	Description
no_blank_line	Removes blank lines on the line above or below.
require_blank_line	Requires a blank line on the line above or below.

```
rule :
  architecture_015:
     style : require_blank_line
```

Warning: It is important to be aware these rules may conflict with rules that enforce rules on previous lines. This can occur when a below rule is applied and then on the next line a previous rule applies. Resolve any conflicts by changing the configuration of either rule.

7.18.1 Example: require_blank_line

The following code would fail with this option:

```
architecture rtl of fifo is
  -- Comment

architecture rtl of fifo is
  signal s_sig1 : std_logic;
```

The following code would pass with this option:

```
architecture rtl of fifo is

  -- Comment

architecture rtl of fifo is

  signal s_sig1 : std_logic;
```

7.18.2 Example: no_blank_line

The following code would fail with this option:

```
architecture rtl of fifo is

  -- Comment

architecture rtl of fifo is

  signal s_sig1 : std_logic;
```

The following code would pass with this option:

```
architecture rtl of fifo is
  -- Comment

architecture rtl of fifo is
  signal s_sig1 : std_logic;
```

7.18.3 Rules Enforcing Blank Lines

- architecture_015
- architecture_016
- architecture_017
- architecture_018
- architecture_200
- block_201
- block_202
- block_203
- block_204
- block_205
- case_008
- case_009
- case_010
- component_018
- context_023
- context_024
- context_025
- function_007
- generate_003
- if_030
- instantiation_019
- package_011
- package_012
- package_body_201

- package_body_202
- package_body_203
- process_011
- process_021
- process_022
- process_023
- process_026
- process_027
- type_011

7.19 Configuring Previous Line Rules

There are rules which will check the contents on lines above code structures. These rules allow enforcement of comments and blank lines.

There are several options to these rules, which can be selected by using the `style` option:

Style	Description
no_blank_line	Removes blank lines on the line above.
require_blank_line	Requires a blank line on the line above.
no_code	Either a blank line; or comment(s) on the line(s) above.
allow_comment	Either a blank line; or comment(s) on the line(s) above and a blank line above the comment(s).
require_comment	Comment(s) required on the line(s) above and a blank line above the comment(s).

Note: Unless stated in the rule description, the default style is `require_blank_line`.

Warning: It is important to be aware these rules may conflict with rules that enforce blank lines below keywords. This can occur when a below rule is applied and then on the next line a previous rule applies. Resolve any conflicts by changing the configuration of either rule.

This is an example of how to configure these options.

```
rule :
  entity_003:
     style : require_blank_line
```

Note: All examples below are using the rule **entity_004**.

7.19.1 Example: no_blank

The following code would fail with this option:

```
library fifo_dsn;
-- Define entity

entity fifo is
```

The following code would pass with this option:

```
library fifo_dsn;
-- Define entity
entity fifo is
```

7.19.2 Example: require_blank_line

The following code would fail with this option:

```
library fifo_dsn;
-- Define entity
entity fifo is
```

The following code would pass with this option:

```
library fifo_dsn;
-- Define entity

entity fifo is
```

7.19.3 Example: no_code

The following code would fail with this option:

```
library fifo_dsn;
entity fifo is
```

The following code would pass with this option:

```
library fifo_dsn;

entity fifo is

library fifo_dsn;
-- Comment

entity fifo is

library fifo_dsn;
-- Comment
entity fifo is
```

7.19.4 Example: allow_comment

The following code would fail with this option:

```
library fifo_dsn;
entity fifo is

library fifo_dsn;
-- Comment
entity fifo is
```

The following code would pass with this option:

```
library fifo_dsn;

entity fifo is

library fifo_dsn;
-- Comment

entity fifo is

library fifo_dsn;

-- Comment
entity fifo is
```

7.19.5 Example: require_comment

The following code would fail these options:

```
library fifo_dsn;
entity fifo is

library fifo_dsn;
-- Comment
entity fifo is
```

The following code would pass these options:

```
library fifo_dsn;

-- Comment
entity fifo is
```

7.19.6 Rules Enforcing Previous Lines

- architecture_003
- block_200
- case_007
- component_003
- context_003
- entity_003
- function_006

- generate_004
- if_031
- instantiation_004
- library_003
- package_003
- package_body_200
- process_015
- type_010

7.20 Configuring Type of Instantiations

There are two methods to instantiate components: component or entity.

VSG can check which method is being used and throw a violation if the incorrect method is detected.

7.20.1 Overriding Type of Instantiation

The default setting is **component** instantiation. We can use the following configuration to change it to **entity** instantiation.

```yaml
---

rule :
  instantiation_034:
     method: 'entity'
```

7.20.2 Rules Enforcing Type of Instantiations

- instantiation_034

7.21 Configuring Optional Items

There are optional language items in VHDL. In the Language Reference Manual (LRM) they are denoted with square brackets []. Using many of these optional items improves the readability of VHDL code.

However, it may not fit the current style of existing code bases. The rules checking the optional items can be configured to add or remove them.

7.21.1 Adding Optional Items

This is the default behavior for these rules.

The configuration format to **add** the optional items is shown below:

```
rule :
  <rule_id>:
     action: 'add'
```

7.21.2 Removing Optional Items

The configuration format to **remove** the optional items is shown below:

```
rule :
  <rule_id>:
     action: 'remove'
```

7.21.3 Rules Enforcing Optional Items

- architecture_010
- architecture_024
- block_002
- block_007
- component_021
- context_021
- context_022
- entity_015
- entity_019
- instantiation_033
- package_007
- package_014
- package_body_002
- package_body_003
- process_012
- process_018

7.22 Configuring Block Comments

Block comments are sequential comment lines with a header and footer. Below are several examples of a block comments:

```
-----------------------------------
-- Comment
-- Comment
-----------------------------------


--================================
```

(continues on next page)

(continued from previous page)

```
-- Comment
-- Comment
--================================

--+-------------------------------
--| Comment
--| Comment
--+-------------------------------

--+---------< header >------------
--| Comment
--| Comment
--+---------< footer >------------
```

7.22.1 Block Comment Structure

The above examples can be generalized into the following:

```
--<header_left><header_left_repeat><header_string><header_right_repeat>
--<comment_left>
--<footer_left><footer_left_repeat><footer_string><footer_right_repeat>
```

Where:

Attribute	Values	Default	Description
header_left	String None	None	The string to place to the right of the –
header_left_repeat	String	-	A character to repeat between header_left and header_string
header_string	String None	None	A string to place in the header.
header_right_repeat	String None	None	A character to repeat after the header-string
comment_left	String None	None	A string which should exist to the right of the –
footer_left	String None	None	The string to place to the right of the –
footer_left_repeat	String	-	A character to repeat between footer_left and footer_string
footer_string	String None	None	A string to place in the footer.
footer_right_repeat	String None	None	A character to repeat after the footer_string

There are additional options for configuring block comments:

Attribute	Values	Default	Description
min_height	Integer	3	Sets minimum number of consecutive comment lines before being considered a block comment.
header_alignment	"left" "center" "right"	"center"	Sets horizontal position of header string.
max_header_column	Integer	120	Sets the maximum length of the combined header.
footer_alignment	"left" "center" "right"	"center"	Sets horizontal position of footer string.
max_footer_column	Integer	120	Sets the maximum length of the combined footer.
allow_indenting	Boolean	True	Allows indented block comments. Setting this to False will only detect block comments starting at column 0.

With these options, a block comment can be validated by VSG.

7.22.2 Examples

It is important to note the rules are disabled by default. They must enabled using a configuration.

Simple Block Comment

To configure the following example…

```
----------------------------------
-- Comment
-- Comment
----------------------------------
```

…the configuration would be:

```
rule:
  block_comment_001:
    disable : False
    header_left : None
    header_left_repeat : '-'
    header_string : None
    header_right_repeat : None
  block_comment_002:
    disable : False
    comment_left : None
  block_comment_003:
    disable : False
    footer_left : None
    footer_left_repeat : '-'
    footer_string : None
    footer_right_repeat : None
```

Complex Block Comment

To configure the following example…

```
--+-<Header>=============================
--| Purpose:
--| Author:
--+-----------------------------<Footer>=
```

…the configuration would be:

```
rule:
  block_comment_001:
    disable : False
    header_left : '+'
    header_left_repeat : '-'
    header_string : '<Header>'
    header_right_repeat : '='
    header_alignment : 'left'
  block_comment_002:
    disable : False
    comment_left : '|'
  block_comment_003:
```

(continues on next page)

(continued from previous page)

```
    disable : False
    footer_left : '+'
    footer_left_repeat : '-'
    footer_string : '<Footer>'
    footer_right_repeat : '='
    footer_alignment : 'right'
```

Doxygen Block Comment

Doxygen comments use an exclamation mark. To configure a block comment for Doxygen. . .

```
----------------------------------
--! Comment
--! Comment
----------------------------------
```

. . . the configuration would be:

```
rule:
  block_comment_001:
    disable : False
    header_left : '-'
    header_left_repeat : '-'
    header_string : None
    header_right_repeat : None
  block_comment_002:
    disable : False
    comment_left : '!'
  block_comment_003:
    disable : False
    footer_left : '-'
    footer_left_repeat : '-'
    footer_string : None
    footer_right_repeat : None
```

7.22.3 Rules Enforcing Block Comments

- block_comment_001
- block_comment_002
- block_comment_003

7.23 Configuring Indentation

VSG follows a predefined set of rules when indenting VHDL code. The indenting alogrithm is driven by a YAML file.

The indent values feeding the algorithm can be obtained by using the **-oc** command line argument. There will be a section starting with **indent**.

7.23.1 Understanding the Indent Configuration Data Structure

The indent configuration file follows this basic format:

```
indent:
    tokens:
        group_name:
            token_name:
                token : value
                after : value
```

where:

Attribute	Values	Description
indent	NA	Indicates the following information defines indent behavior.
tokens	NA	Indicates the following information defines token level behavior.
group_name	<string>	The group a token belongs to.
token_name	<string>	The name of the token which has indent behavior.
token	NA	Indicates the value to apply to the token.
after	NA	Indicates the value to apply after the token.
value	**<integer>** current "+<integer>" "-<integer>"	The type of behavior to apply to the token or after the token.

The **group_name** and **token_name** keys provide unique identifier which can be matched to types of tokens after the file has been parsed. There are more tokens than are currently defined in the indent configuration, as not all tokens require indenting rules.

The **token** key informs VSG how to apply indents when it encounters the token.

The **after** key informs VSG how to apply indents to successive tokens it encounters.

The **value** defines the behavior for each **token** and **after** key, and are defined as:

Value	Type	Description
[0-9][0-9]*	<integer>	Sets the indent level to the specified value.
current	<string>	Uses the existing indent level.
"+[0-9][0-9]*"	<string>	Increase the indent relative to the current indent level.
"-[0-9][0-9]*"	<string>	Decrease the indent relative to the current indent level.

Using the **group_name** and **token_name** to identify types of VHDL tokens and then the **token** and **after** defines the behavior of the indenting algorithm.

7.23.2 Example

VSG assumes the closing parenthesis will match with the **port** keyword.

```
entity some_block is
  port (
    I_CLK    : std_logic;
    I_RST    : std_logic;
    I_WR_EN  : std_logic;
    O_DATA   : std_logic_vector(7 downto 0);
  );
end entity some_block;
```

If we use the following configuration…

```
indent:
    tokens:
        port_clause:
            close_parenthesis:
                token : current
                after : '-2'
```

…then VSG will enforce the following format:

```
entity some_block is
  port (
    I_CLK    : std_logic;
    I_RST    : std_logic;
    I_WR_EN  : std_logic;
    O_DATA   : std_logic_vector(7 downto 0);
    );
end entity some_block;
```

How does this work?

VSG is setting the indent levels as it goes. The port definitions in the above example are set to an indent of 2. When the closing parenthesis is encountered, VSG checks the **port_clause.close_parenthesis.token** key to determine what to do. In this case the key is set to **current**. This tells VSG to keep the indent of 2 for the closing parenthesis token. VSG then looks at the **port_clause.close_parenthesis.after** key and finds a **'-2'**. This tells VSG to subtract two from the current indent value of 2. Which will set the indent to 0. The next token in the indent configuration with a **token** key value of **current** would then get 0.

7.23.3 The Challenge With Adjusting Indent Values

The most difficult part of changing the indent values is knowing which **group_name** and **token_name** to use.

For the **group_name** use the VHDL LRM as a reference. All group names match a *left-hand side* of a *production*.

For the **token_name**, refer to the output configuration using the **-oc**. This will give the complete indent configuration. The desired adjustment can be pulled out into a smaller file. This file can then be applied with the **-c** option.

7.24 Configuring Multiline Indent Rules

There are rules which will check indent of multiline expressions and conditions.

There are several options to these rules:

Method	Type	Default	Description
align_left	boolean	True	True = New lines will be aligned left. False = Align to left of assignment operator.
align_paren	boolean	True	True = Use open parenthesis for alignment. False = Do not use open parenthesis for alignment.

This is an example of how to configure the option.

```
rule :
  constant_012:
     align_left : False
     align_paren : True
```

Note: All examples below are using the rule **constant_012**.

7.24.1 Example: align_left True, align_paren False

The following code would fail with this option:

```
constant c_const : t_type :=
                             (
                               (
                                 a => 0,
                                 b => 1
                               ),
                               (
                                 c => 0,
                                 d => 1
                               )
                             );

constant c_const : t_type :=
(
 (
  a => 0,
  b => 1
 ),
 (
  c => 0,
  d => 1
 )
);
```

The following code would pass with this option:

```
constant c_const : t_type :=
(
  (
    a => 0,
    b => 1
  ),
  (
    c => 0,
```

(continues on next page)

(continued from previous page)

```
    d => 1
  )
);

constant c_const : t_type :=
(
  (
    a => 0,
    b => 1
  ),
  (
    c => 0,
    d => 1
  )
);
```

7.24.2 Example: align_left False, align_paren False

The following code would fail with this option:

```
constant c_const : t_type :=
(
  (
    a => 0,
    b => 1
  ),
  (
    c => 0,
    d => 1
  )
);
```

The following code would pass with this option:

```
constant c_const : t_type :=
                             (
                               (
                                 a => 0,
                                 b => 1
                               ),
                               (
                                 c => 0,
                                 d => 1
                               )
                             );
```

7.24.3 Example: align_left True, align_paren True

The following code would fail with this option:

```
constant c_const : t_type := (
  1 => func1(
    G_GENERIC1, G_GENERIC2)
);
```

The following code would pass with this option:

```
constant c_const : t_type := (
  1 => func1(
               G_GENERIC1, G_GENERIC2)
);
```

7.24.4 Rules Enforcing Multiline Indent Rules

- concurrent_003
- if_004
- process_020
- sequential_004
- variable_assignment_004

7.25 Configuring Multiline Structure Rules

There are rules which will check indent and formatting of multiline expressions and conditions.

The alignment of multiline rules is handled by a corresponding rule. Both rules are required to ensure proper formatting of multiline expressions and conditions. The corresponding rule will be noted in the rule documentation.

There are several options to these rules:

Method	Type	Default	Description
first_paren_new_line	string	yes	First opening parenthesis on it's own line.
last_paren_new_line	string	yes	Last closing parenthesis on it's own line.
open_paren_new_line	string	yes	Insert new line after open parenthesis.
close_paren_new_line	string	yes	Insert new line before close parenthesis.
new_line_after_comma	string	yes	Insert new line after the commas.
assign_on_single_line	string	yes	Keep assignments on a single line.
ignore_single_line	string	yes	Do not apply rules if expression/condition is contained on a single line.
move_last_comment	string	ignore	If last_paren_new_line is 'yes', then move any trailing comments to the previous line.

The options can be combined to format the output.

Each option except `new_line_after_comma` and `assign_on_single_line` allows one of three values: yes, no and ignore.

Option Value	Action
yes	Option will be enforced.
no	The inverse of the Option will be enforced.
ignore	The option will be ignored.

The `new_line_after_comma` option allows one of four values: yes, no, ignore and ignore_positional.

Option Value	Action
yes	Insert new line after commas.
no	Remove new line after commas.
ignore	Ignore commas.
ignore_positional	Insert new line after commas unless elements are positional.

The `assign_on_single_line` option allows one of two values: yes and ignore.

Option Value	Action
yes	Force assignments to a single line.
ignore	Allow assignments to span multiple lines.

This is an example of how to configure these options.

```
rule :
  constant_012:
     first_paren_new_line : 'yes'
     last_paren_new_line : 'yes'
     open_paren_new_line : 'yes'
     close_paren_new_line : 'yes'
     new_line_after_comma : 'ignore'
     ignore_single_line : 'no'
```

Note: All examples below are using the rule **constant_016** and the option ignore_single_line is False.

7.25.1 Example: first_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := (a => 0, b => 1);
```

The following code would pass with this option:

```
constant c_const : t_type :=
(a => 0, b => 1);
```

7.25.2 Example: last_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := (a => 0, b => 1);
```

The following code would pass with this option:

```
constant c_const : t_type := (a => 0, b => 1
);
```

7.25.3 Example: first_paren_new_line and last_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := (a => 0, b => 1);
```

The following code would pass with this option:

```
constant c_const : t_type :=
(
  a => 0, b => 1
);
```

7.25.4 Example: new_line_after_comma

The following code would fail with this option:

```
constant c_const : t_type := (a => 0, b => 1);
```

The following code would pass with this option:

```
constant c_const : t_type := (a => 0,
b => 1);
```

7.25.5 Example: new_line_after_comma and first_paren_new_line and last_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := (a => 0, b => 1);
```

The following code would pass with this option:

```
constant c_const : t_type :=
(a => 0,
 b => 1);
```

7.25.6 Example: open_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := ((a => 0, b => 1), (c => 0, d => 1));
```

The following code would pass with this option:

```
constant c_const : t_type := (
(
  a => 0, b => 1), (
c => 0, d => 1));
```

7.25.7 Example: close_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := ((a => 0, b => 1), (c => 0, d => 1));
```

The following code would pass with this option:

```
constant c_const : t_type := ((a => 0, b => 1
                              ), (c => 0, d => 1
                              ));
```

7.25.8 Example: open_paren_new_line and close_paren_new_line

The following code would fail with this option:

```
constant c_const : t_type := ((a => 0, b => 1), (c => 0, d => 1));
```

The following code would pass with this option:

```
constant c_const : t_type := (
(
  a => 0, b => 1
), (
  c => 0, d => 1
));
```

7.25.9 Example: all options yes

The following code would fail with this option:

```
constant c_const : t_type := ((a => 0, b => 1), (c => 0, d => 1));
```

The following code would pass with this option:

```
constant c_const : t_type :=
(
  (
    a => 0,
    b => 1
  ),
  (
    c => 0,
    d => 1
  )
);
```

7.25.10 Example: all options no

The following code would pass with this option:

```
constant c_const : t_type :=
(
  (
    a => 0,
    b => 1
  ),
  (
    c => 0,
    d => 1
  )
);
```

The following code would fail with this option:

```
constant c_const : t_type := ((a => 0, b => 1), (c => 0, d => 1));
```

7.25.11 Example: assign_on_single_line

The following code would pass with this option set to True:

```
constant c_const : t_type :=
(
  1 => func1(std_logic_vector(G_GEN), G_GEN2),
  2 => func1(std_logic_vector(G_GEN), G_GEN2)
);
```

The following code would fail with this option set to True:

```
constant c_const : t_type :=
(
  1 => func1(std_logic_vector(G_GEN), G_GEN2),
  2 => func1(
          std_logic_vector(G_GEN), G_GEN2)
);
```

7.25.12 Example: last_paren_new_line and move_last_comment

The following code would fail with this option:

```
constant c_const : t_type :=
(
  a => 0,
  b => 1); -- Comment
```

The following code would pass with this option:

```
constant c_const : t_type :=
(
  a => 0,
  b => 1 -- Comment
);
```

7.25.13 Rules Enforcing Multiline Structure Rules

- concurrent_011

7.26 Configuring Concurrent Alignment Rules

There are rules which will check indent and alignment of multiline conditional expressions and conditional waveforms.

Conditional expressions and conditional waveforms are defined as:

```
conditional_expressions ::=
  expression **when** condition
  { **else** expression **when** condition }
  [ **else** expression ]

conditional_waveforms ::=
  waveform **when** condition
  { **else** waveform **when** condition }
  [ **else** waveform ]
```

Below is an example of a conditional waveform:

```
architecture rtl of fifo is

begin

  output <= '1' when input = "00" else
            sig_a or sig_b when input = "01" else
            sig_c and sig_d when input = "10" else
            '0';

end architecture rtl;
```

The alignment of multiline rules is handled by a corresponding rule. Both rules are required to ensure proper formatting of multiline expressions and conditions. The corresponding rule will be noted in the rule documentation.

There are several options to these rules:

Option	Type	Default	Description
align_left	string	'no'	Align multilines to the left.
align_paren	string	'yes'	Indent lines based on parenthesis.
align_when_keywords	string	'no'	Each when keyword will be aligned.
wrap_at_when	string	'yes'	Indent multiline condition at 'when' keyword.
align_else_keywords	string	'no'	Each else keyword will be aligned.

The options can be combined to format the conditional expression or conditional waveform.

Each option allows one of two values: 'yes' and 'no'.

Option Value	Action
'yes'	Option will be enforced.
'no'	The inverse of the Option will be enforced.

This is an example of how to configure these options.

```
rule :
  concurrent_009:
     wrap_at_when : 'yes'
     align_when_keywords : 'yes'
     align_else_keywords : 'yes'
     align_left : 'no'
```

Note: All examples below are using the rule **concurrent_009**.

7.26.1 Example: indent_condition_at_when

The following code would fail with this option:

```
output <= '1' when input = "0000" or
    input = "1111" else
  sig_a or sig_b when input = "0001" and
    input = "1001" else
  sig_c and sig_d when input = "0010" or
    input = "1010" else
  '0';
```

The following code would pass with this option:

```
output <= '1' when input = "0000" or
                   input = "1111" else
  sig_a or sig_b when input = "0001" and
                      input = "1001" else
  sig_c and sig_d when input = "0010" or
                       input = "1010" else
  '0';
```

7.26.2 Example: align_when_keywords

The following code would fail with this option:

```
output <= '1' when input = "00" else
          sig_a or sig_b when input = "01" else
          sig_c and sig_d when input = "10" else
          '0';
```

The following code would pass with this option:

```
output <= '1'             when input = "00" else
          sig_a or sig_b  when input = "01" else
          sig_c and sig_d when input = "10" else
          '0';
```

7.26.3 Example: align_when_keywords and align_else_keywords

The following code would fail with this option:

```
output <= '1' when input = "0000" else
          sig_a or sig_b when input = "0100" and input = "1100" else
          sig_c when input = "10" else
          '0';
```

The following code would pass with this option:

```
output <= '1'            when input = "0000"                     else
          sig_a or sig_b when input = "0100" and input = "1100" else
          sig_c          when input = "10"                       else
          '0';
```

7.26.4 Example: align_left 'yes'

The following code would fail with this option:

```
output <= '1' when input = "0000" else
          sig_a or sig_b when input = "0100" and input = "1100" else
          sig_c when input = "10" else
          '0';
```

The following code would pass with this option:

```
output <= '1' when input = "0000" else
  sig_a or sig_b when input = "0100" and input = "1100" else
  sig_c when input = "10" else
  '0';
```

7.26.5 Example: align_left 'no'

The following code would fail with this option:

```
output <= '1' when input = "0000" else
  sig_a or sig_b when input = "0100" and input = "1100" else
  sig_c when input = "10" else
  '0';
```

The following code would pass with this option:

```
output <= '1' when input = "0000" else
          sig_a or sig_b when input = "0100" and input = "1100" else
          sig_c when input = "10" else
          '0';
```

7.26.6 Example: align_paren 'yes' and align_left 'no'

The following code would fail with this option:

```
output <= '1' when func1(func2(G_VALUE1,
                    G_VALUE2), func3(
                    G_VALUE3)
                    ) else
          '0';
```

The following code would pass with this option:

```
output <= '1' when func1(func2(G_VALUE1,
                               G_VALUE2), func3(
                                                G_VALUE3)
                        ) else
          '0';
```

7.26.7 Rules Enforcing Conditional Expression

- concurrent_009

7.27 Configuring Concurrent Structure Rules

There are rules which will check the structure of conditional expressions and waveforms.

The alignment of multiline rules is handled by a corresponding rule. Both rules are required to ensure proper formatting of multiline conditional expressions and waveforms. The corresponding rule will be noted in the rule documentation.

There are several options to these rules:

Method	Type	Default	Description
new_line_after_assign	string	no	First opening parenthesis on it's own line.
ignore_single_line	string	yes	Do not apply rules if expression/condition is contained on a single line.

The options can be combined to format the output.

Each option except `ignore_single_line` allows one of three values: yes, no and ignore.

Option Value	Action
yes	Option will be enforced.
no	The inverse of the Option will be enforced.
ignore	The option will be ignored.

The `ignore_single_line` option allows one of two values: yes and ignore.

Option Value	Action
yes	Force assignments to a single line.
ignore	Allow assignments to span multiple lines.

This is an example of how to configure these options.

```
rule :
  concurrent_011:
     ignore_single_line : 'no'
```

Note: All examples below are using the rule **concurrent_011** and the option ignore_single_line is 'no'.

7.27.1 Example: new_line_after_assign

The following code would fail with this option:

```
write_en <= '1' when sig1 = "00" else '0';
```

The following code would pass with this option:

```
write_en <=
  '1' when sig1 = "00" else '0';
```

7.27.2 Rules Enforcing Conditional Expression Structure

- concurrent_011

CHAPTER 8

Code Tags

VSG supports inline tags embedded into code to enable or disable rules. This can be useful in fine tuning rule exceptions within a file. The code tags are embedded in comments similar to pragmas, and must be on it's own line.

8.1 Full rule exclusion

Entire portions of a file can be ignored using the **vsg_off** and **vsg_on** tags.

```
-- vsg_off
process (write, read, full) is
begin
  a <= write;
  b <= read;
end process;
-- vsg_on
```

The **vsg_off** tag disables all rule checking. The **vsg_on** tag enables all rule checking, except those disabled by a configuration.

8.2 Individual Rule Exclusions

Individual rules can be disabled by adding the rule identifier to the **vsg_off** and **vsg_on** tags. Multiple identifiers can be added.

```
-- vsg_off process_016 process_018
process (write, read, full) is
begin
  a <= write;
  b <= read;
end process;
-- vsg_on
```

The bare **vsg_on** enables all rules not disabled by a configuration.

Each rule can be independently enabled or disabled:

```
-- vsg_off process_016 process_018
process (write, read, full) is
begin
  a <= write;
  b <= read;
end process;

-- vsg_on process_016
FIFO_PROC : process (write, read, full) is
begin
  a <= write;
  b <= read;
end process;

-- vsg_on process_018
FIFO_PROC : process (write, read, full) is
begin
  a <= write;
  b <= read;
end process FIFO_PROC;
```

In the previous example, the *process_016* and *process_018* are disabled for the first process. *Process_018* is disabled for the second process. No rules are disabled for the third process.

8.3 Next Line Rule Exclusions

Rules can be disabled for a single line using the **vsg_disable_next_line** tag. Multiple identifiers can be added to a single tag..

```
-- vsg_disable_next_line process_016
process (write, read, full) is
begin
  a <= write;
  b <= read;
  -- vsg_disable_next_line process_018
end process;
```

In the above example, process_016 will only be disabled for the line with the process keyword. Successive processes without labels will be flagged by process_016.

Sequential next line exclusions will also be honored:

```
-- vsg_disable_next_line process_002
-- vsg_disable_next_line process_016
process(write, read, full) is
begin
  a <= write;
  b <= read;
  -- vsg_disable_next_line process_018
end process;
```

In the above example, both process_002 and process_016 will be disabled for the line starting with the process keyword.

CHAPTER 9

Editor Integration

If your editor can execute programs on the command line, you can run VSG without having to leave your editor. This brings a new level of efficiency to coding in VHDL.

9.1 VIM

Add the following macro into your .vimrc file:

```
map <F9> :setl autoread<CR>:let b:current_file = @%<CR>:w!<CR>:execute '!vsg -f ' . b:current_file ' --fix'<CR><CR>:edit<CR>:setl noautoread<CR>
```

This macro bound to the <F9> key performs the following steps:

1. Save the current buffer
2. Execute vsg with the –fix option
3. Reload the buffer

When you are editing a file, you can hit <F9> and VSG will run on the current buffer without leaving VIM.

CHAPTER 10

Tool Integration

VSG supports integration with other tools via several command line options.

–all-phases	Executes all phases without stopping if a violation is found.
–json	Filename of JSON file to generate.
–fix_only	Filename of JSON file with fix instructions

10.1 –all-phases

VSG has a concept of phases, where violations in one phase should be addressed before moving to the next phase. The **–all-phases** option will run an analysis over all the phases. It will not stop if a violation has occured.

This option can be useful when integrating VSG into an editor that supports linters. It is important to note there are dependencies between some rules. If violations for a later phase are fixed before violations on an earlier phase, it could lead to reoccurances of violations until the correct order is followed.

10.2 –json

The violations discovered by VSG can be saved in a JSON formatted file. This eases the transferring information from VSG to another tool.

Below is the basic format of the JSON file:

```
{
  "<filename>": {
    "violations": [
      {
        "rule": <rule_id>,
        "linenumber": <linenumber>,
        "solution": <solution>
```

(continues on next page)

(continued from previous page)

```
      }
    ]
  }
}
```

where:

<filename>	Name of the file violations refer to.
<rule_id>	The rule identifier and number.
<linenumber>	The linenumber of the violation.
<solution>	The solution required to fix the violation.

10.3 –fix_only

Using this option with the **–fix** option will restrict the rules fixed base on a JSON file. This allows tools a finer granularity in instructing VSG how to fix a file.

Below is the basic format of the JSON file:

```
{
  "fix": {
     "rule": {
       "<rule_id>": [ <number> ]
     }
  }
}
```

where:

<rule_id>	The rule identifier and number.
<number>	If this value is "all", then all violations will be fixed. If it is a series of numbers, then only those lines will be fixed.

It is important to note there are rules that will modify the line number at which errors occur. The number reported at the command line or via the **–json** option are after all rules have been applied. Therefore, when using **–fix_only** option the line numbers given in the JSON file may not line up with the line number while VSG is analyzing the file while it is being modified.

CHAPTER 11

Pragmas

VSG treats all pragmas as comments. Most pragmas are ignored as they do not affect the style of the code.

However, the following pragmas do affect the parser: **–vhdl_comp_off** and **–vhdl_comp_on**.

With these pragmas, it is possible to write code that would not follow the VHDL Language Reference Manual (LRM).

Take the following code as an example:

```
--vhdl_comp_off
entity FIFO is
--vhdl_comp_on

entity FIFO is
end entity;
```

A parser which did not take the pragmas into account would fail because the code would appear to the parser as:

```
entity FIFO is

entity FIFO is
end entity;
```

Which does not follow the VHDL LRM.

When VSG encounters the **–vhdl_comp_off** pragma, it will ignore anything after it until it encounters the **–vhdl_comp_on** pragma. No formatting will be enforced between the pragmas.

CHAPTER 12

Localizing

Warning: This information is out of date with the release of 3.0.0

Warning: This information is out of date with the release of 3.0.0

Note: If you have local rules defined in a version prior to 3.0.0, create an issue and I can work with you to convert it to 3.0.0 format.

VSG supports customization to your coding style standards by allowing localized rules. These rules are stored in a directory with an __init__.py file and one or more python files. The files should follow the same structure and naming convention as the rules found in the vsg/rules directory.

The localized rules will be used when the **–local_rules** command line argument is given or using the **local_rules** option in a configuration file.

12.1 Example: Create rule to check for entity and architectures in the same file.

Let's suppose in our organization the entity and architecture should be split into separate files. This rule is not in the base rule set, but we can add it through localization. For this example, we will be setting up the localized rules in your home directory.

12.1.1 Prepare local rules directory

Create an empty directory with an empty __init__.py file

```
$ mkdir ~/local_rules
$ touch ~/local_rules/__init__.py
```

12.1.2 Create new rule file

We will create a new rule by extending the base rule class.

Note: The file name and class name must start with **rule_**. Otherwise VSG will not recognize it as a rule.

The rule will be in the **localized** group. Since this is the first rule, we will number it **001**.

```
from vsg import rule


class rule_001(rule.rule):

  def __init__(self):
      rule.rule.__init__(self, 'localized', '001')
```

Referencing the *Phases*, we decide it should be in phase 1: structural.

```
from vsg import rule


class rule_001(rule.rule):

  def __init__(self):
      rule.rule.__init__(self, 'localized', '001')
      self.phase = 1
```

Now we need to add the **analyze** method to perform the check.

```
from vsg import rule


class rule_001(rule.rule):

  def __init__(self):
      rule.rule.__init__(self, 'localized', '001')
      self.phase = 1

  def analyze(self, oFile):
```

The built in variables in the vsg.line class can be used to build rules. In this case, the vsg.vhdlFile class has two attributes (**hasEntity** and **hasArchitecture**) that are exactly what we need. We are ready to write the body of the **analyze** method:

```
from vsg import rule


class rule_001(rule.rule):

  def __init__(self):
      rule.rule.__init__(self, 'localized', '001')
```

(continues on next page)

(continued from previous page)

```
        self.phase = 1

    def analyze(self, oFile):
        if oFile.hasEntity and oFile.hasArchitecture:
            self.add_violation(utils.create_violation_dict(1))
```

The base rule class has an **add_violation** method which takes a dictionary as an argument. The *create_violation_dict* function will create the dictionary. This dictionary can be modified to include other information about the violation. This method appends the dictionary to a violation list, which is processed later for reporting and fixing purposes. In this case, any line number will do so we picked 1.

We must decide if we want to give VSG the ability to fix this rule on it's own. If so, then we will need to write the **_fix_violations** method. However, for this violation we want the user to split the file. We will tell VSG the rule is not fixable.

```
from vsg import rule


class rule_001(rule.rule):

    def __init__(self):
        rule.rule.__init__(self, 'localized', '001')
        self.phase = 1
        self.fixable = False  # User must split the file

    def analyze(self, oFile):
        if oFile.hasEntity and oFile.hasArchitecture:
            self.add_violation(utils.create_violation_dict(1))
```

We also need to provide a solution to the user so they will know how to fix the violation:

```
from vsg import rule


class rule_001(rule.rule):

    def __init__(self):
        rule.rule.__init__(self, 'localized', '001')
        self.phase = 1


        self.fixable = False  # User must split the file
        self.solution = 'Split entity and architecture into seperate files.'

    def analyze(self, oFile):
        if oFile.hasEntity and oFile.hasArchitecture:
            self.add_violation(utils.create_violation_dict(1))
```

Finally, we need to add a code tag check so the rule can be disabled via comments in the code:

```
from vsg import rule


class rule_001(rule.rule):

    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        rule.rule.__init__(self, 'localized', '001')
        self.phase = 1
        self.fixable = False  # User must split the file
        self.solution = 'Split entity and architecture into seperate files.'

    def analyze(self, oFile):
        if not self.is_vsg_off(oLine):
            if oFile.hasEntity and oFile.hasArchitecture:
                self.add_violation(utils.create_violation_dict(1))
```

The rule is complete, so we save it as rule_localized_001.py. Performing an **ls** on our local_rules directory:

```
$ ls ~/local_rules
__init__.py  rule_localized_001.py
```

12.1.3 Use new rule to analyze

When we want to run with localized rules, use the **–local_rules** option.

```
$ vsg -f RAM.vhd --local_rules ~/local_rules
File:  RAM.vhd
==============
Phase 1... Reporting
localized_001            |            1 | Split entity and architecture into seperate␣
↪files.
Phase 2... Not executed
Phase 3... Not executed
Phase 4... Not executed
Phase 5... Not executed
Phase 6... Not executed
Phase 7... Not executed
==============
Total Rules Checked: 50
Total Failures:      1
```

Our new rule will now flag files which have both an entity and an architecture in the same file. That was a fairly simple rule. To write more complex rules, it is important to understand how the rule class works.

12.2 Understanding the Rule class

Every rule uses the base rule class. There are a few methods to the base rule class, but we are interested in only the following:

Method	Description
add_violations	Adds violations to a list.
analyze	Calls _pre_analyze and then _analyze.
_analyze	Code that performs the analysis.
fix	calls analyze and then _fix_violations.
_fix_violations	Code that fixes the violations.
_get_solution	Prints out the solution to stdout.
_pre_analyze	Code that sets up variables for _analyze.

We will look at the rule **constant_014** to illustrate how VSG uses the methods above:

```
class rule_014(rule.rule):
    '''
    Constant rule 014 checks the indent of multiline constants that are not arrays.
    '''

    def __init__(self):
        rule.rule.__init__(self)
        self.name = 'constant'
        self.identifier = '014'
        self.solution = 'Align with := keyword on constant declaration line.'
        self.phase = 5

    def _pre_analyze(self):
        self.alignmentColumn = 0
        self.fKeywordFound = False

    def _analyze(self, oFile, oLine, iLineNumber):
        if not oLine.isConstantArray and oLine.insideConstant:
            if oLine.isConstant and ':=' in oLine.line:
                self.alignmentColumn = oLine.line.index(':=') + len(':= ')
                self.fKeywordFound = True
            elif not oLine.isConstant and self.fKeywordFound:
                sMatch = ' ' * self.alignmentColumn
                if not re.match('^' + sMatch + '\w', oLine.line):
                    self.add_violation(utils.create_violation_dict(LineNumber))
                    self.dFix['violations'][iLineNumber] = self.alignmentColumn
            if oLine.isConstantEnd:
                self.fKeywordFound = False

    def _fix_violations(self, oFile):
        for iLineNumber in self.violations:
            sLine = oFile.lines[iLineNumber].line
            sNewLine = ' ' * self.dFix['violations'][iLineNumber] + sLine.strip()
            oFile.lines[iLineNumber].update_line(sNewLine)
```

12.2.1 Creating Class

First we create the rule by inheriting from the base rule class. We also add a comment to describe what the rule is doing.

```
class rule_014(rule.rule):
    '''
    Constant rule 014 checks the indent of multiline constants that are not arrays.
    '''
```

12.2.2 Adding __init__

Then we add the **__init__** method. It calls the init of the base rule class, then we modify attributes for this specific rule:

```
def __init__(self):
    rule.rule.__init__(self)
```

(continues on next page)

(continued from previous page)

```
    self.name = 'constant'
    self.identifier = '014'
    self.solution = 'Align with := keyword on constant declaration line.'
    self.phase = 5
```

For this rule we set it's *name*, *identifier*, *solution*, and *phase*.

12.2.3 Analyzing Considerations

The **analyze** method of the base rule class will first call **_pre_anaylze** before **_analyze**. The **_analyze** method is wrapped in a loop that increments through each line of the file. The **analyze** method also checks if the rule has been turned off for a line, via code tags. If the code tag indicates to ignore the line, then it will be skipped. If you decide to override the **analyze** method, then you should add the code tag check.

12.2.4 Adding _pre_analyze method

In this rule, we use the **_pre_analyze** method to initialize some variables. These variables must be set outside the loop that is present in the **analyze** method.

```
def _pre_analyze(self):
    self.alignmentColumn = 0
    self.fKeywordFound = False
```

12.2.5 Adding _analyze method

The **_analyze** method is called on every line of the VHDL file. Any memory needed between lines must be declared in the **_pre_analyze** method. In the following code, notice *self.alignmentColumn* and *self.fKeywordFound*.

```
def _analyze(self, oFile, oLine, iLineNumber):
    if not oLine.isConstantArray and oLine.insideConstant:
        if oLine.isConstant and ':=' in oLine.line:
            self.alignmentColumn = oLine.line.index(':=') + len(':= ')
            self.fKeywordFound = True
        elif not oLine.isConstant and self.fKeywordFound:
            sMatch = ' ' * self.alignmentColumn
            if not re.match('^' + sMatch + '\w', oLine.line):
                self.add_violation(utils.create_violation_dict(LineNumber))
                self.dFix['violations'][iLineNumber] = self.alignmentColumn
        if oLine.isConstantEnd:
            self.fKeywordFound = False
```

This code is searching for the characteristics of a non-array constant.

```
def _analyze(self, oFile, oLine, iLineNumber):
    if not oLine.isConstantArray and oLine.insideConstant:
```

Once the non-array constant is found, it notes the column of the *:=* keyword.

```
if oLine.isConstant and ':=' in oLine.line:
    self.alignmentColumn = oLine.line.index(':=') + len(':= ')
    self.fKeywordFound = True
```

On successive lines of the constant declaration, it checks to see if there are enough spaces from the beginning of the line to match the column number the *:=* is located at.

```
elif not oLine.isConstant and self.fKeywordFound:
```

If there are not enough spaces, then a violation is added. We also store off the required column into a predefined dictionary named *dFix*. This will be used later when the **fix** method is called.

```
sMatch = ' ' * self.alignmentColumn
if not re.match('^' + sMatch + '\w', oLine.line):
    self.add_violation(utils.create_violation_dict(LineNumber))
    self.dFix['violations'][iLineNumber] = self.alignmentColumn
```

When we detect the end of the constant declaration, we clear a flag and prepare for the next constant declaration.

```
if oLine.isConstantEnd:
    self.fKeywordFound = False
```

12.2.6 Fixing considerations

The **fix** method will first call the **analyze** method and then the **_fix_violations** method. Unlike the **analyze** method, it does not wrap the **_fix_violations** in a loop. This is due to some fixes needing to execute either top down or bottom up. Rules that add or delete lines need to work from the bottom up. Otherwise, the violations detected by the **analyze** method will have moved.

12.2.7 Adding the _fix_violations method

In this rule, we are going to iterate on all the violations in the *self.violations* attribute.

```
def _fix_violations(self, oFile):
    for iLineNumber in self.violations:
```

We store the current line off to make it easier to read. Then we strip the line of all leading and trailing spaces and prepend the number of spaces required to align with the *:=* keyword.

```
sLine = oFile.lines[iLineNumber].line
sNewLine = ' ' * self.dFix['violations'][iLineNumber] + sLine.strip()
```

Finally, we update the line with our modified line using the **update_line** method.

```
oFile.lines[iLineNumber].update_line(sNewLine)
```

12.3 Violation dictionary

Violations are stored as a list of dictionaries in the **rule.violations** attribute. This is the generic format of the dictionary represented by json:

```
{
  "lines" : [
     {
       "number" : "<integer>",
```

(continues on next page)

(continued from previous page)

```
        "<line_attribute>" : "<line_value>",
        "<line_attribute>" : "<line_value>"
      }
    ],
  "<violation_attribute>" : "<violation_value>",
  "<violation_attribute>" : "<violation_value>"
}
```

This format gives us the greatest flexibility in describing violations. The lines[0][‘number’] is the only required element in a violation dictionary. The “<line_attribute>” and “<violation_attribute>” elements are optional. They are used by more complex rules to maintain information used to fix violations.

12.3.1 Single line violations

Most violations are against a single line and no other information is required to fix it. These dictionaries use the minimumal form.

```
{
  "lines" : [
      {
        "number" : 40
      }
    ]
}
```

12.3.2 Single line violations with additional information

If additional information for single line violations is required, it will be stored at the **violation** level.

```
{
  "lines" : [
    {
      "number" : 40
    }
  ],
  "label" : "FIFO"
}
```

This violation is indicating there is an issue at line 40 with the label “FIFO”. The “label” element will be used to fix the violation.

12.3.3 Multiple line violations

If a rule covers multiple lines, then information about individual lines can be stored:

```
{
  "lines" : [
    {
      "number" : 40,
      "column" : 20
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "number" : 41,
      "column" : 35
    }
  ],
  "desired_column" : 15
}
```

In the above case, we are trying to align a keyword over multiple lines. Each line which is not aligned is reported in the **lines** list. The **column** attribute indicates which column the keyword was found. The **desired_column**, which applies to all lines in the **lines** list, indicates which column the keyword should be located.

This violation would cover a group of multiple lines. If there were violations in multiple groups, then each group with get it's own violation dictionary.

12.3.4 utils functions

There are three functions in the utils module to help with managing the violation dictionary: **create_violation_dict**, **get_violation_line_number** and **get_violating_line**. The **create_violation_dict** will return a dictionary in the form of the single line violation described above. Use this to create the initial violation and add to it as necessary.

The **get_violation_line_number** will return the lines['number'] attribute of the violation. Use this function to abstract away the line number from the underlying data structure.

The **get_violating_line** will return a line object at the line the violation occured. This is easier than manually indexing into the oFile list to pull out a line.

12.4 Rule creation guidelines

Keep these points in mind when creating new rules:

1. Use an existing rule as a starting point
2. Remember that **analyze** calls **_pre_analyze** and then **_analyze**
3. Override **_get_solution** to return complex messages
4. **analyze** method can be overridden if necessary
5. If overriding **analyze**, then include a check for *vsg_off*

CHAPTER 13

Phases

Rules are grouped together and executed in phases. This simplifies rule generation for rules in later phases. If issues are found during a phase, then successive phases will not be run. The phases are constructed to model the proper order of fixing issues. Each phase prepares the code for the next phase.

Which phase a rule is executed in is indicated by one of these phase labels:

13.1 Phase - Structural

This ensures the VHDL is structured properly for future phases.

It includes the following operations:

- Addition or removal of optional VHDL elements
- Addition or carriage returns to split lines
- Removal of carriage returns to combine lines

13.2 Phase - Whitespace

This phase checks whitespace rules.

It includes the following operations:

- Addition of whitespace between VHDL elements
- Reduction of whitespace between VHDL elements

13.3 Phase - Vertical Spacing

This phase checks vertical spacing between lines.

It includes the following operations:

- Addition of carriage returns to emphasize VHDL elements
- Removal of carriage returns to deemphasize VHDL elements

13.4 Phase - Indentation

This phase checks the indent of lines.

13.5 Phase - Alignment

This phase checks VHDL elements are column aligned.

It includes the following operations:

- Alignment of colons
- Alignment of assignment operators
- Alignment of identifiers

13.6 Phase - Capitalization

This phase checks case of VHDL elements.

It includes the following operations:

- Case of VHDL keywords
- Case of identifiers

13.7 Phase - Naming conventions

This phase checks naming conventions for non VHDL keywords.

It includes the following operations:

- Signal prefixes
- Port prefixes and suffixes
- Architecture identifiers

13.8 Subphases

Each phase can have multiple subphases. There are rules which are executed within the same phase, but one is dependent on another. Utilizing a subphase allows for the proper execution of the rules.

13.8.1 Subphase 1

Prepare code for rules in subphase 2.

13.8.2 Subphase 2

Execute on code prepared in subphase 1.

CHAPTER 14

Rule Severity

VSG supports rule severity with two built in levels: Error and Warning. The default behavior for most rules is **Error**. Only the **Error** severity level will result in an exit status of 1. **Errors** will also be the only errors written to a JUnit XML file if that option is chosen.

The severity level of each rule is indicated with one of the following icons in the rule description:

Warning :

Error:

The table below summarizes the built-in severities:

Name	Type	Exit Status	JUnit	Syntastic	Description
Error	error	1	Yes	ERROR	A rule which must be fixed.
Warning	warning	0	No	WARNING	A rule which should be fixed.

14.1 Configuring Severity Levels

The existing severity level of a rule can be configured. For example, if you want to change the line length rule, *length_001*, to an **Error** instead of a **Warning**, use the following configuration:

```
{
    "rule":{
        "length_001":{
            "severity": "Error"
        }
    }
}
```

14.2 Defining Severity Levels

VSG supports user defined severity level. Any new severity level will follow the same rules as the severity it is based on. It will be reported to the screen, but will not be reported in JUnit XML files and will not force an exit status of 1.

To create your own severity level, create a configuration which defines just the severity level following this format:

```
{
    "severity":{
        "Future":{
            "type":"warning"
        },
        "Todo":{
            "type":"error"
        }
    }
}
```

This configuration defines two new severities: **Future** and **Todo**. The **Future** severity is set to the **warning** type. The **Todo** severity is set to the **error** type.

The newly defined severity levels can then be applied to a rule using a second configuration.

```
{
    "rule":{
        "length_001":{
            "severity": "Future"
        }
    }
}
```

Apply the defined severity levels by calling both configurations:

```
vsg -c severity.json rule_configuration.json -f fifo.vhd
```

14.2.1 Rules Which are Warnings by Default

- *length_001*
- *length_002*
- *length_003*

CHAPTER 15

Rules

The rules are divided into catagories depending on the part of the VHDL code being operated on.

15.1 After Rules

15.1.1 after_001

This rule checks for **after x** in signal assignments in clock processes.

Violation

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0';
    b <= '1';
  elsif (clock'event and clock = '1') then
    a <= d;
    b <= c;
  end if;
end process clk_proc;
```

Fix

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0';
    b <= '1';
  elsif (clock'event and clock = '1') then
    a <= d after 1 ns;
```

(continues on next page)

(continued from previous page)

```
    b <= c after 1 ns;
  end if;
end process clk_proc;
```

Note: This rule has two configurable items:

- magnitude
- units

The **magnitude** is the number of units. Default is *1*.

The **units** is a valid time unit: ms, us, ns, ps etc... Default is *ns*.

15.1.2 after_002

This rule checks the *after* keywords are aligned in a clock process. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0';
    b <= '1';
  elsif (clock'event and clock = '1') then
    a <= d      after 1 ns;
    b <= c    after 1 ns;
  end if;
end process clk_proc;
```

Fix

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0';
    b <= '1';
  elsif (clock'event and clock = '1') then
    a <= d      after 1 ns;
    b <= c      after 1 ns;
  end if;
end process clk_proc;
```

15.1.3 after_003

This rule checks the *after* keywords do not exist in the reset portion of a clock process.

Violation

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0' after 1 ns;
    b <= '1' after 1 ns;
  elsif (clock'event and clock = '1') then
    a <= d after 1 ns;
    b <= c after 1 ns;
  end if;
end process clk_proc;
```

Fix

```
clk_proc : process(clock, reset) is
begin
  if (reset = '1') then
    a <= '0';
    b <= '1';
  elsif (clock'event and clock = '1') then
    a <= d  after 1 ns;
    b <= c  after 1 ns;
  end if;
end process clk_proc;
```

15.2 Architecture Rules

15.2.1 architecture_001

This rule checks for blank spaces before the **architecture** keyword.

Violation

```
  architecture rtl of fifo is
begin
```

Fix

```
architecture rtl of fifo is
begin
```

15.2.2 architecture_002

This rule has been split into the following rules:

- *architecture_030*
- *architecture_031*
- *architecture_032*
- *architecture_033*

15.2.3 architecture_003

This rule checks for a blank lines or comments above the **architecture** declaration.

Refer to Configuring Previous Line Rules for options.

Violation

```
library ieee;
architecture rtl of fifo is
```

Fix

```
library ieee;

architecture rtl of fifo is
```

15.2.4 architecture_004

This rule checks the proper case of the **architecture** keyword in the architecture declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ARCHITECTURE rtl of fifo is
```

```
architecture rtl of fifo is
```

15.2.5 architecture_005

This rule checks the **of** keyword is on the same line as the **architecture** keyword.

Violation

```
architecture rtl
  of fifo is
```

Fix

```
architecture rtl of
fifo is
```

15.2.6 architecture_006

This rule checks the **is** keyword is on the same line as the **architecture** keyword.

Violation

```
architecture rtl of fifo
  is

architecture rtl of fifo
```

Fix

```
architecture rtl of fifo is

architecture rtl of fifo is
```

15.2.7 architecture_007

This rule checks for spaces before the **begin** keyword.

Violation

```
architecture rtl of fifo is
  begin
```

Fix

```
architecture rtl of fifo is
begin
```

15.2.8 architecture_008

This rule checks for spaces before the **end architecture** keywords.

Violation

```
architecture rtl of fifo is
begin
  end architecture
```

Fix

```
architecture rtl of fifo is
begin
end architecture
```

15.2.9 architecture_009

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END architecture;

End architecture;
```

Fix

```
end architecture;

end architecture;
```

15.2.10 architecture_010

This rule checks for the keyword **architecture** in the **end architecture** statement. It is clearer to the reader to state what is ending.

Refer to the section Configuring Optional Items for options.

Violation

```
end architecture_name;
```

Fix

```
end architecture architecture_name;
```

15.2.11 architecture_011

This rule checks the architecture name case in the **end architecture** statement.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end architecture ARCHITECTURE_NAME;
```

Fix

```
end architecture architecture_name;
```

15.2.12 architecture_012

This rule checks for a single space between **end** and **architecture** keywords.

Violation

```
end    architecture architecture_name;
```

Fix

```
end architecture architecture_name;
```

15.2.13 architecture_013

This rule checks the case of the architecture name in the architecture declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture RTL of fifo is
```

Fix

```
architecture rtl of fifo is
```

15.2.14 architecture_014

This rule checks the case of the entity name in the architecture declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture rtl of FIFO is
```

Fix

```
architecture rtl of fifo is
```

15.2.15 architecture_015

This rule checks for blank lines below the architecture declaration.

Refer to Configuring Blank Lines for options.

Violation

```
architecture rtl of fifo is
  signal wr_en : std_logic;
begin
```

Fix

```
architecture rtl of fifo is

  signal wr_en : std_logic;
begin
```

15.2.16 architecture_016

This rule checks for blank lines above the **begin** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
architecture rtl of fifo is

  signal wr_en : std_logic;
begin
```

Fix

```
architecture rtl of fifo is

  signal wr_en : std_logic;

begin
```

15.2.17 architecture_017

This rule checks for a blank line below the **begin** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
begin
  wr_en <= '0';
```

Fix

```
begin

  wr_en <= '0';
```

15.2.18 architecture_018

This rule checks for blank lines or comments above the **end architecture** declaration.

Refer to Configuring Blank Lines for options.

Violation

```
  rd_en <= '1';
end architecture RTL;
```

Fix

```
  rd_en <= '1';

end architecture RTL;
```

15.2.19 architecture_019

This rule checks the proper case of the **of** keyword in the architecture declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture rtl OF fifo is
```

Fix

```
architecture rtl of fifo is
```

15.2.20 architecture_020

This rule checks the proper case of the **is** keyword in the architecture declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture rtl of fifo IS
```

Fix

```
architecture rtl of fifo is
```

15.2.21 architecture_021

This rule checks the proper case of the **begin** keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture rtl of fifo is
BEGIN
```

Fix

```
architecture rtl of fifo is
begin
```

15.2.22 architecture_022

This rule checks for a single space before the entity name in the end architecture declaration.

Violation

```
end architecture    fifo;
```

Fix

```
end architecture fifo;
```

15.2.23 architecture_024

This rule checks for the architecture name in the **end architecture** statement. It is clearer to the reader to state which architecture the end statement is closing.

Refer to the section Configuring Optional Items for options.

Violation

```
end architecture;
```

Fix

```
end architecture architecture_name;
```

15.2.24 architecture_025

This rule checks for valid names for the architecture. Typical architecture names are: RTL, EMPTY, and BEHAVE. This rule allows the user to restrict what can be used for an architecture name.

Note: This rule is disabled by default. You can enable and configure the names using the following configuration.

```
---

rule :
  architecture_025 :
    disabled : False
    names :
      - rtl
      - empty
      - behave
```

Violation

```
architecture some_invalid_arch_name of entity1 is
```

Fix

The user is required to decide which is the correct architecture name.

15.2.25 architecture_026

This rule checks the colons are in the same column for all declarations in the architecture declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
architecture rtl of my_entity is

  signal   wr_en : std_logic;
  signal   rd_en    : std_logic;
  constant c_period : time;

begin
```

Fix

```
architecture rtl of my_entity is

  signal   wr_en    : std_logic;
  signal   rd_en    : std_logic;
  constant c_period : time;

begin
```

15.2.26 architecture_027

This rule checks the alignment of inline comments in the architecture declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
architecture rtl of my_entity is

  signal   wr_en    : std_logic;  -- Comment 1
  signal   rd_en    : std_logic;      -- Comment 2
  constant c_period : time; -- Comment 3

begin
```

Fix

```
architecture rtl of my_entity is

  signal   wr_en    : std_logic; -- Comment 1
  signal   rd_en    : std_logic; -- Comment 2
  constant c_period : time;      -- Comment 3
```

(continues on next page)

(continued from previous page)

```
begin
```

15.2.27 architecture_028

This rule checks the **architecture** keyword in the **end architecture** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end ARCHITECTURE;

end Architecture;
```

Fix

```
end architecture;

end architecture;
```

15.2.28 architecture_029

This rule checks for alignment of identifiers in attribute, type, subtype, constant, signal, variable and file declarations in the architecture declarative region.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
signal    sig1 : std_logic;
file some_file :
variable v_var1 : std_logic;
type t_myType : std_logic;
```

Fix

```
signal   sig1 : std_logic;
file     some_file :
variable v_var1 : std_logic;
type     t_myType : std_logic;
```

15.2.29 architecture_030

This rule checks for a single space between **architecture** and the identifier.

Violation

```
architecture    rtl of fifo is
```

Fix

```
architecture rtl of fifo is
```

15.2.30 architecture_031

This rule checks for a single space between the identifier and the **of** keyword.

Violation

```
architecture rtl    of fifo is
```

Fix

```
architecture rtl of fifo is
```

15.2.31 architecture_032

This rule checks for a single space between the **of** keyword and the entity_name.

Violation

```
architecture rtl of    fifo is
```

Fix

```
architecture rtl of fifo is
```

15.2.32 architecture_033

This rule checks for a single space between the entity_name and the **is** keyword.

Violation

```
architecture rtl of fifo    is
```

Fix

```
architecture rtl of fifo is
```

15.2.33 architecture_200

This rule checks for a blank line below the end architecture statement.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end architecture;
library ieee;
```

Fix

```
end architecture;

library ieee;
```

15.2.34 architecture_600

This rule checks for consistent capitalization of generic names in an architecture body.

Violation

```
entity FIFO is
  generic (
    G_WIDTH : natural := 16
  );
end entity fifo;

architecture rtl of fifo is

   signal w_data : std_logic_vector(g_width - 1 downto 0);

begin

   output <= large_data(g_width - 1 downto 0);

end architecture rtl;
```

Fix

```
entity FIFO is
  generic (
    G_WIDTH : natural := 16
  );
end entity fifo;

architecture rtl of fifo is

   signal w_data : std_logic_vector(G_WIDTH - 1 downto 0);

begin

   output <= large_data(G_WIDTH - 1 downto 0);

end architecture rtl;
```

15.3 Assert Rules

15.3.1 assert_001

This rule checks indent of multiline assert statements.

Violation

```
assert WIDTH > 16
     report "FIFO width is limited to 16 bits."
 severity FAILURE;
```

Fix

```
assert WIDTH > 16
  report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.3.2 assert_002

This rule checks the **report** keyword is on it's own line for concurrent assertion statements.

Violation

```
architecture rtl of fifo is
begin

  assert WIDTH > 16 report "FIFO width is limited to 16 bits."
    severity FAILURE;

end architecture rtl;
```

Fix

```
architecture rtl of fifo is
begin

  assert WIDTH > 16
    report "FIFO width is limited to 16 bits."
    severity FAILURE;

end architecture rtl;
```

15.3.3 assert_003

This rule checks the **report** keyword is on it's own line for assertion statements.

Violation

```
architecture rtl of fifo is
begin

```

(continues on next page)

(continued from previous page)

```
  process
  begin

    assert WIDTH > 16 report "FIFO width is limited to 16 bits."
      severity FAILURE;

  end process;

end architecture rtl;
```

Fix

```
architecture rtl of fifo is
begin

  process
  begin

    assert WIDTH > 16
      report "FIFO width is limited to 16 bits."
      severity FAILURE;

  end process;

end architecture rtl;
```

15.3.4 assert_004

This rule checks the **severity** keyword is on it's own line for concurrent assertion statements.

Violation

```
architecture rtl of fifo is
begin

  assert WIDTH > 16
    report "FIFO width is limited to 16 bits." severity FAILURE;

end architecture rtl;
```

Fix

```
architecture rtl of fifo is
begin

  assert WIDTH > 16
    report "FIFO width is limited to 16 bits."
    severity FAILURE;

end architecture rtl;
```

15.3.5 assert_400

This rule checks the alignment of the report expressions.

Note: There is a configuration option **alignment** which changes the indent location of multiple lines.

alignment set to 'report' (Default)

Violation

```
assert WIDTH > 16
  report "FIFO width is limited" &
" to 16 bits."
  severity FAILURE;
```

Fix

```
assert WIDTH > 16
  report "FIFO width is limited" &
         " to 16 bits."
  severity FAILURE;
```

alignment set to 'left'

Violation

```
assert WIDTH > 16
  report "FIFO width is limited" &
" to 16 bits."
  severity FAILURE;
```

Fix

```
assert WIDTH > 16
  report "FIFO width is limited" &
    " to 16 bits."
  severity FAILURE;
```

15.4 Attribute Rules

15.4.1 attribute_001

This rule has been superceeded by:

- attribute_declaration_300
- attribute_specification_300

15.4.2 attribute_002

This rule has been superceeded by:

- attribute_declaration_500

- attribute_specification_500

15.5 Attribute Declaration Rules

15.5.1 attribute_declaration_100

This rule checks for a single space after the following elements: **attribute** keyword and colon.

Violation

```
attribute    max_delay :    time;
```

Fix

```
attribute max_delay : time;
```

15.5.2 attribute_declaration_101

This rule checks for at least a single space before the colon.

Violation

```
attribute max_delay: time;
```

Fix

```
attribute max_delay : time;
```

15.5.3 attribute_declaration_300

This rule checks the indent of the **attribute** keyword.

Violation

```
signal sig1 : std_logic;
   attribute max_delay : time;
```

Fix

```
signal sig1 : std_logic;
attribute max_delay : time;
```

15.5.4 attribute_declaration_500

This rule checks the **attribute** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ATTRIBUTE max_delay : time;
```

Fix

```
attribute max_delay : time;
```

15.5.5 attribute_declaration_501

This rule checks the *identifier* has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute MAX_DELAY : time;
```

Fix

```
attribute max_delay : time;
```

15.5.6 attribute_declaration_502

This rule checks the *type_mark* has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute max_delay : TIME;
```

Fix

```
attribute max_delay : time;
```

15.6 Attribute Specification Rules

15.6.1 attribute_specification_100

This rule checks for a single space after the following attribute_specification elements: **attribute** keyword, *attribute_designator*, **of** keyword and **is** keyword.

Violation

```
attribute   coordinate   of   comp_1:component   is   (0.0, 17.5);

attribute   coordinate   of   comp_1:component   is(0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1:component   is (0.0, 17.5);

attribute coordinate of comp_1:component   is (0.0, 17.5);
```

15.6.2 attribute_specification_101

This rule checks for a single space before the **is** keyword.

Violation

```
attribute coordinate of comp_1 : component   is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);

attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.6.3 attribute_specification_300

This rule checks the indent of the **attribute** keyword.

Violation

```
signal sig1 : std_logic;
   attribute coordinate of comp_1 : component is (0.0, 17.5);
```

Fix

```
signal sig1 : std_logic;
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.6.4 attribute_specification_500

This rule checks the **attribute** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ATTRIBUTE coordinate of comp_1 : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.6.5 attribute_specification_501

This rule checks the *attribute_designator* has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute COORDINATE of comp_1 : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.6.6 attribute_specification_502

This rule checks the **of** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate OF comp_1 : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.6.7 attribute_specification_503

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate of comp_1 : component IS (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.7 Block Rules

15.7.1 block_001

This rule checks the block label and the **block** keyword are on the same line. Keeping the label and generate on the same line reduces excessive indenting.

Violation

```
block_label :
  block is
```

Fix

```
block_label : block is
```

15.7.2 block_002

This rule checks for the existence of the **is** keyword.

Refer to the section Configuring Optional Items for options.

Violation

```
block_label : block
block_label : block (guard_condition)
```

Fix

```
block_label : block is
block_label : block (guard_condition) is
```

15.7.3 block_003

This rule checks the **is** keyword is on the same line as the **block** keyword.

Violation

```
block_label : block
is
```

Fix

```
block_labeel : block is
```

15.7.4 block_004

This rule checks the **begin** keyword is on it's own line.

Violation

```
block is begin
```

Fix

```
block is
begin
```

15.7.5 block_005

This rule checks for code after the **begin** keyword.

Violation

```
begin a <= b;
```

Fix

```
begin
a <= b;
```

15.7.6 block_006

This rule checks the **end** keyword is on it's own line.

Violation

```
a <= b; end block;
```

Fix

```
a <= b;
end block;
```

15.7.7 block_007

This rule checks the block label exists in the closing of the block statement.

Refer to the section Configuring Optional Items for options.

Violation

```
end block;
```

Fix

```
end block block_label;
```

15.7.8 block_100

This rule checks for a single space between the following block elements: label, label colon, **block** keyword, guard open parenthesis, guart close parenthesis, and **is** keywords.

Violation

```
block_label    :     block     (guard_condition)    is
block_label   :    block     is
```

Fix

```
block_label : block (guard_condition) is
block_label : block is
```

15.7.9 block_101

This rule checks for a single space between the **end** and **block** keywords and label.

Violation

```
end   block    block_label;
```

Fix

```
end block block_label;
```

15.7.10 block_200

This rule checks for blank lines or comments above the block label.

Refer to Configuring Previous Line Rules for options.

Violation

```
a <= b;
block_label : block is
```

Fix

```
a <= b;

block_label : block is
```

15.7.11 block_201

This rule checks for a blank line below the **block** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
block_label : block is
  constant width : integer := 32;
```

Fix

```
block_label : block is

  constant width : integer := 32;
```

15.7.12 block_202

This rule checks for blank lines or comments above the **begin** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
block_label block is

  constant width : integer := 32;
begin
```

Fix

```
block_label block is

  constant width : integer := 32;

begin
```

15.7.13 block_203

This rule checks for a blank line below the **begin** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
begin
  a <= b;
```

Fix

```
begin

  a <= b;
```

15.7.14 block_204

This rule checks for blank lines or comments above the **end** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
begin

  a <= b;
end block block_label;
```

Fix

```
begin

  a <= b;

end block block_label;
```

15.7.15 block_205

This rule checks for a blank line below the semicolon.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end block block_label;
a <= b;
```

Fix

```
end block block_label;

a <= b;
```

15.7.16 block_300

This rule checks the indent of the block label.

Violation

```
a <= b;

   block_label : block is
```

Fix

```
a <= b;

block_label : block is
```

15.7.17 block_301

This rule checks the indent of the **begin** keyword.

Violation

```
block_label : block is

  begin
```

Fix

```
block_label : block is

begin
```

15.7.18 block_302

This rule checks the indent of the **end** keyword.

Violation

```
block_label : block is

begin

  end block block_label;
```

Fix

```
block_label : block is

begin

end block block_label;
```

15.7.19 block_400

This rule checks the identifiers for all declarations are aligned in the block declarative region.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable    var1 : natural;
constant  c_period : time;
```

Fix

```
variable var1     : natural;
constant c_period : time;
```

15.7.20 block_401

This rule checks the colons are in the same column for all declarations in the block declarative part. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
signal sig1: natural;
variable var2  : natural;
constant c_period : time;
file my_test_input : my_file_type;
```

Fix

```
signal sig1          : natural;
variable var2        : natural;
constant c_period    : time;
file my_test_input : my_file_type;
```

15.7.21 block_500

This rule checks the label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
BLOCK_LABEL : block is
```

Fix

```
block_label : block is
```

15.7.22 block_501

This rule checks the **block** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
block_label : BLOCK is
```

Fix

```
block_label : block is
```

15.7.23 block_502

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
block_label : block IS
```

Fix

```
block_label : block is
```

15.7.24 block_503

This rule checks the **begin** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
block_label : block is
BEGIN
```

Fix

```
block_label : block is
begin
```

15.7.25 block_504

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END block block_label;
```

Fix

```
end block block_label;
```

15.7.26 block_505

This rule checks the **block** keyword in the **end block** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end BLOCK block_label;
```

Fix

```
end block block_label;
```

15.7.27 block_506

This rule checks the label has proper case on the end block declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end block BLOCK_LABEL;
```

Fix

```
end block block_label;
```

15.7.28 block_600

This rule checks for valid suffixes on block labels. The default suffix is *_blk*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
block_label : block is
```

Fix

```
block_label_blk : block is
```

15.7.29 block_601

This rule checks for valid prefixes on block labels. The default prefix is *blk_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
block_label : block is
```

Fix

```
blk_block_label : block is
```

15.8 Block Comment Rules

Note: All examples in this section are using the following options:

- header_left = '+'
- header_left_repeat = '-'
- header_string = '[Header]'
- header_right_repeat = '='
- comment_left = '|'
- footer_left = '+'
- footer_left_repeat = '-'
- footer_string = '[Footer]'
- footer_right_repeat = '='
- min_height = 3
- header_alignment = 'center'
- max_header_column = 40
- footer_alignment = 'right'
- max_footer_column = 40

15.8.1 block_comment_001

This rule checks the block comment header is correct.

Refer to the section Configuring Block Comments for additional information.

Violation

```
----------------------------------------
--   Comment
--   Comment
----------------------------------------
```

Fix

```
--+-------------[ Header ]==============
--   Comment
--   Comment
----------------------------------------
```

15.8.2 block_comment_002

This rule checks the **comment_left** attribute exists for all comments.

Refer to the section Configuring Block Comments for additional information.

Violation

```
--+-------------[ Header ]==============
--   Comment
--   Comment
----------------------------------------
```

Fix

```
--+-------------[ Header ]==============
--|  Comment
--|  Comment
----------------------------------------
```

15.8.3 block_comment_003

This rule checks the block comment footer is correct.

Refer to the section Configuring Block Comments for additional information.

Violation

```
--+-------------[ Header ]==============
--|  Comment
--|  Comment
----------------------------------------
```

Fix

```
--+-------------[ Header ]==============
--|  Comment
--|  Comment
--+-------------------------[ Footer ]=
```

15.9 Case Rules

15.9.1 case_001

This rule checks the indent of **case**, **when**, and **end case** keywords.

Violation

```
 case data is

     when 0 =>
 when 1 =>
         when 3 =>

end case;
```

Fix

```
case data is

  when 0 =>
  when 1 =>
  when 3 =>

end case;
```

15.9.2 case_002

This rule checks for a single space after the **case** keyword.

Violation

```
case    data is
```

Fix

```
case data is
```

15.9.3 case_003

This rule checks for a single space before the **is** keyword.

Violation

```
case data    is
```

Fix

```
case data is
```

15.9.4 case_004

This rule checks for a single space after the **when** keyword.

Violation

```
case data is

  when    3 =>
```

Fix

```
case data is

  when 3 =>
```

15.9.5 case_005

This rule checks for a single space before the **=>** operator.

Violation

```
case data is

  when 3    =>
```

Fix

```
case data is

  when 3 =>
```

15.9.6 case_006

This rule checks for a single space between the **end** and **case** keywords.

Violation

```
case data is

end     case;
```

Fix

```
case data is

end case;
```

15.9.7 case_007

This rule checks for blank lines or comments above the **case** keyword.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
a <= '1';
case data is


-- This is a comment
case data is
```

Fix

```
a <= '1';

case data is


-- This is a comment
case data is
```

15.9.8 case_008

This rule checks for a blank line below the **is** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
case data is
  when 0 =>
```

Fix

```
case data is

  when 0 =>
```

15.9.9 case_009

This rule checks for blank lines or comments above the **end** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
  when others =>
    null;
end case;
```

Fix

```
  when others =>
    null;

end case;
```

15.9.10 case_010

This rule checks for a blank line below the **end case** keywords.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end case;
a <= '1';
```

Fix

```
end case;

a <= '1';
```

15.9.11 case_011

This rule checks the alignment of multiline **when** statements.

Violation

```
case data is

  when 0 | 1 | 2 | 3
   4 | 5 | 7 =>
```

Fix

```
case data is

  when 0 | 1 | 2 | 3
       4 | 5 | 7 =>
```

15.9.12 case_012

This rule checks for code after the **=>** operator.

Violation

```
when 0 => a <= '1';
```

Fix

```
when 0 =>
  a <= '1';
```

15.9.13 case_013

This rule checks the indent of the **null** keyword.

Violation

```
  when others =>
     null;

  when others =>
null;
```

Fix

```
when others =>
  null;

when others =>
  null;
```

15.9.14 case_014

This rule checks the **case** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
CASE address is

Case address is

case address is
```

Fix

```
case address is

case address is

case address is
```

15.9.15 case_015

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
case address IS

case address Is

case address iS
```

Fix

```
case address is

case address is

case address is
```

15.9.16 case_016

This rule checks the **when** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
WHEN a =>
When b =>
when c =>
```

Fix

```
when a =>
when b =>
when c =>
```

15.9.17 case_017

This rule checks the **end** keyword in the **end case** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
End case;
END case;
end case;
```

Fix

```
end case;
end case;
end case;
```

15.9.18 case_018

This rule checks the **case** keyword has proper case in the **end case**.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end CASE;
end CAse;
end case;
```

Fix

```
end case;
end case;
end case;
```

15.9.19 case_019

This rule checks for labels before the **case** keyword. The label should be removed. The preference is to have comments above the case statement.

Violation

```
CASE_LABEL : case address is
CASE_LABEL: case address is
case address is
```

Fix

```
case address is
case address is
case address is
```

15.9.20 case_020

This rule checks for labels after the **end case** keywords. The label should be removed. The preference is to have comments above the case statement.

Violation

```
end case CASE_LABEL;
end case;
```

Fix

```
end case;
end case;
```

15.9.21 case_021

This rule aligns consecutive comment only lines above a **when** keyword in a case statement with the **when** keyword.

Violation

```
      -- comment 1
-- comment 2
   -- comment 3
  when wr_en =>
    rd_en <= '0';
```

Fix

```
-- comment 1
-- comment 2
-- comment 3
when wr_en =>
  rd_en <= '0';
```

15.10 Comment Rules

15.10.1 comment_004

This rule checks for at least a single space before inline comments.

Violation

```
wr_en <= '1';--Write data
rd_en <= '1';    -- Read data
```

Fix

```
wr_en <= '1'; --Write data
rd_en <= '1';    -- Read data
```

15.10.2 comment_010

This rule checks the indent lines starting with comments.

Violation

```
    -- Libraries
libary ieee;

  -- Define architecture
architecture rtl of fifo is

-- Define signals
  signal wr_en : std_logic;
  signal rd_en : std_Logic;

begin
```

Fix

```
-- Libraries
libary ieee;

-- Define architecture
architecture rtl of fifo is

  -- Define signals
  signal wr_en : std_logic;
  signal rd_en : std_Logic;

begin
```

15.10.3 comment_011

This rule checks for in-line comments and moves them to the line above.

Note: This rule is disabled by default.

Violation

```
a <= b; -- Assign signal
```

Fix

```
-- Assign signal
a <= b;
```

15.11 Component Rules

15.11.1 component_001

This rule checks the indentation of the **component** keyword.

Violation

```
architecture rtl of fifo is
begin

component fifo is

     component ram is
```

Fix

```
architecture rtl of fifo is
begin

  component fifo is

  component ram is
```

15.11.2 component_002

This rule checks for a single space after the **component** keyword.

Violation

```
component    fifo is
```

Fix

```
component fifo is
```

15.11.3 component_003

This rule checks for blank lines or comments above the **component** declaration.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
end component fifo;
component ram is
```

Fix

```
end component fifo;

component ram is
```

15.11.4 component_004

This rule checks the **component** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
COMPONENT fifo is

Component fifo is
```

Fix

```
component fifo is

component fifo is
```

15.11.5 component_005

This rule checks the **is** keyword is on the same line as the **component** keyword.

Violation

```
component fifo

component fifo
is
```

Fix

```
component fifo is

component fifo is
```

15.11.6 component_006

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
component fifo IS

component fifo Is
```

Fix

```
component fifo is

component fifo is
```

15.11.7 component_007

This rule checks for a single space before the **is** keyword.

Violation

```
component fifo    is
```

Fix

```
component fifo is
```

15.11.8 component_008

This rule checks the component name has proper case in the component declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
component FIFO is
```

Fix

```
component fifo is
```

15.11.9 component_009

This rule checks the indent of the **end component** keywords.

Violation

```
  overflow : std_logic
);
    end component fifo;
```

Fix

```
    overflow : std_logic
  );
end component fifo;
```

15.11.10 component_010

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END component fifo;
```

Fix

```
end component fifo;
```

15.11.11 component_011

This rule checks for single space after the **end** keyword.

Violation

```
end   component fifo;
```

Fix

```
end component fifo;
```

15.11.12 component_012

This rule checks the proper case of the component name in the **end component** line.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end component FIFO;
```

Fix

```
end component fifo;
```

15.11.13 component_013

This rule checks for a single space after the **component** keyword in the **end component** line.

Violation

```
end component    fifo;
```

Fix

```
end component fifo;
```

15.11.14 component_014

This rule checks the **component** keyword in the **end component** line has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end COMPONENT fifo;
```

Fix

```
end component fifo;
```

15.11.15 component_015

This rule has been depricated. The **component** keyword is required per the LRM.

15.11.16 component_016

This rule checks for blank lines above the **end component** line.

Violation

```
    overflow : std_logic
  );



end component fifo;
```

Fix

```
    overflow : std_logic
  );
end component fifo;
```

15.11.17 component_017

This rule checks the alignment of the colon for each generic and port in the component declaration.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
component my_component
    generic (
        g_width : positive;
        g_output_delay : positive
    );
    port (
        clk_i : in std_logic;
        data_i : in std_logic;
        data_o : in std_logic
    );
end component;
```

Fix

```
component my_component
    generic (
        g_width        : positive;
        g_output_delay : positive
    );
    port (
        clk_i  : in std_logic;
        data_i : in std_logic;
        data_o : in std_logic
    );
end component;
```

15.11.18 component_018

This rule checks for a blank line below the **end component** line.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end component fifo;
signal rd_en : std_logic;
```

Fix

```
end component fifo;

signal rd_en : std_logic;
```

15.11.19 component_019

This rule checks for comments at the end of the port and generic clauses in component declarations. These comments represent additional maintainence. They will be out of sync with the entity at some point. Refer to the entity for port types, port directions and purpose.

Violation

```
wr_en : in    std_logic;  -- Enables write to RAM
rd_en : out   std_logic; -- Enable reads from RAM
```

Fix

```
wr_en : in    std_logic;
rd_en : out   std_logic;
```

15.11.20 component_020

This rule checks for alignment of inline comments in the component declaration.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
component my_component
    generic (
        g_width        : positive;  -- Data width
        g_output_delay : positive -- Delay at output
    );
    port (
        clk_i  : in std_logic; -- Input clock
        data_i : in std_logic;    -- Data input
        data_o : in std_logic -- Data output
    );
end my_component;
```

Fix

```
component my_component
    generic (
        g_width        : positive; -- Data width
        g_output_delay : positive  -- Delay at output
    );
    port (
        clk_i  : in std_logic; -- Input clock
        data_i : in std_logic; -- Data input
        data_o : in std_logic  -- Data output
    );
end my_component;
```

15.11.21 component_021

This rule inserts the optional **is** keyword if it does not exist.

Refer to the section Configuring Optional Items for options.

Violation

```
component my_component

end my_component;
```

Fix

```
component my_component is

end my_component;
```

15.12 Concurrent Rules

15.12.1 concurrent_001

This rule checks the indent of concurrent assignments.

Violation

```
architecture RTL of FIFO is
begin

     wr_en <= '0';
rd_en <= '1';
```

Fix

```
architecture RTL of FIFO is
begin

  wr_en <= '0';
  rd_en <= '1';
```

15.12.2 concurrent_002

This rule checks for a single space after the **<=** operator.

Violation

```
wr_en <=    '0';
rd_en <=   '1';
```

Fix

```
wr_en <= '0';
rd_en <= '1';
```

15.12.3 concurrent_003

This rule checks alignment of multiline concurrent simple signal assignments. Succesive lines should align to the space after the assignment operator. However, there is a special case if there are parenthesis in the assignment. If the parenthesis are not closed on the same line, then the next line will be aligned to the parenthesis. Aligning to the parenthesis improves readability.

Violation

```
O_FOO <= (1 => q_foo(63 downto 32),
         0 => q_foo(31 downto  0));

n_foo <= resize(unsigned(I_FOO) +
         unsigned(I_BAR), q_foo'length);
```

Fix

```
O_FOO <= (1 => q_foo(63 downto 32),
          0 => q_foo(31 downto  0));

n_foo <= resize(unsigned(I_FOO) +
                unsigned(I_BAR), q_foo'length);
```

15.12.4 concurrent_004

This rule checks for at least a single space before the **<=** operator.

Violation

```
wr_en<= '0';
```

Fix

```
wr_en <= '0';
```

15.12.5 concurrent_005

This rule checks for labels on concurrent assignments. Labels on concurrents are optional and do not provide additional information.

Violation

```
WR_EN_OUTPUT : WR_EN <= q_wr_en;
RD_EN_OUTPUT : RD_EN <= q_rd_en;
```

Fix

```
WR_EN <= q_wr_en;
RD_EN <= q_rd_en;
```

15.12.6 concurrent_006

This rule checks the alignment of the **<=** operator over multiple consecutive lines. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
wr_en <= '0';
rd_en   <= '1';
data <= (others => '0');
```

Fix

```
wr_en <= '0';
rd_en <= '1';
data  <= (others => '0');
```

15.12.7 concurrent_007

This rule checks for code after the **else** keyword.

Note: There is a configuration option **allow_single_line** which allows single line concurrent statements.

allow_single_line set to False (Default)

Violation

```
wr_en <= '0' when overflow = '0' else '1';
wr_en <= '0' when overflow = '0' else '1' when underflow = '1' else sig_a;
```

Fix

```
wr_en <= '0' when overflow = '0' else
         '1';
wr_en <= '0' when overflow = '0' else
         '1' when underflow = '1' else
         sig_a;
```

allow_single_line set to True

Violation

```
wr_en <= '0' when overflow = '0' else '1';
wr_en <= '0' when overflow = '0' else '1' when underflow = '1' else sig_a;
```

Fix

```
wr_en <= '0' when overflow = '0' else '1';
wr_en <= '0' when overflow = '0' else
         '1' when underflow = '1' else
         sig_a;
```

15.12.8 concurrent_008

This rule checks the alignment of inline comments in consecutive concurrent statements. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
wr_en <= '0';      -- Write enable
rd_en <= '1';    -- Read enable
data  <= (others => '0'); -- Write data
```

Fix

```
wr_en <= '0';             -- Write enable
rd_en <= '1';             -- Read enable
data  <= (others => '0'); -- Write data
```

15.12.9 concurrent_009

This rule checks alignment of multiline concurrent conditional signal statements.

Refer to the section Configuring Concurrent Alignment Rules for information on formatting options.

Violation

```
wr_en <= '0' when q_wr_en = '1' else
     '1';

w_foo <= I_FOO when ((I_BAR = '1') and
         (I_CRUFT = '1')) else
         '0';
```

Fix

```
wr_en <= '0' when q_wr_en = '1' else
         '1';

w_foo <= I_FOO when ((I_BAR = '1') and
                     (I_CRUFT = '1')) else
         '0';
```

15.12.10 concurrent_010

This rule removes blank lines within concurrent signal assignments.

Violation

```
wr_en <= '0' when q_wr_en = '1' else

     '1';
```

(continues on next page)

(continued from previous page)

```
w_foo <= I_FOO when ((I_BAR = '1') and

                     (I_CRUFT = '1')) else

         '0';
```

Fix

```
wr_en <= '0' when q_wr_en = '1' else
         '1';

w_foo <= I_FOO when ((I_BAR = '1') and
                     (I_CRUFT = '1')) else
         '0';
```

15.12.11 concurrent_011

This rule checks the structure of simple and conditional concurrent statements.

Refer to the section Configuring Concurrent Structure Rules for information on formatting options.

Violation

```
wr_en <=
  '0' when q_wr_en = '1' else
         '1';

w_foo <=
  I_FOO when ((I_BAR = '1') and
                     (I_CRUFT = '1')) else
         '0';
```

Fix

```
wr_en <= '0' when q_wr_en = '1' else
         '1';

w_foo <= I_FOO when ((I_BAR = '1') and
                     (I_CRUFT = '1')) else
         '0';
```

15.13 Constant Rules

15.13.1 constant_001

This rule checks the indent of a constant declaration.

Violation

```
architecture RTL of FIFO is

constant size : integer := 1;
    constant width : integer := 32
```

Fix

```
architecture RTL of FIFO is

  constant size : integer := 1;
  constant width : integer := 32
```

15.13.2 constant_002

This rule checks the **constant** keyword is has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
CONSTANT size : integer := 1;
```

Fix

```
constant size : integer := 1;
```

15.13.3 constant_003

This rule was depricated and replaced with rules: function_015, package_019, procedure_010, architecture_029 and process_037.

15.13.4 constant_004

This rule checks the constant identifier has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
constant SIZE : integer := 1;
```

Fix

```
constant size : integer := 1;
```

15.13.5 constant_005

This rule checks for a single space after the colon.

Violation

```
constant size   :integer := 1;
constant wdith :      integer := 32;
```

Fix

```
constant size  : integer := 1;
constant width : integer := 32;
```

15.13.6 constant_006

This rule checks for at least a single space before the colon.

Violation

```
constant size: integer := 1;
constant width     : integer := 32;
```

Fix

```
constant size : integer := 1;
constant width     : integer := 32;
```

15.13.7 constant_007

This rule checks the **:=** is on the same line at the **constant** keyword.

Violation

```
constant size : integer
   := 1;
```

Fix

```
constant size : integer := 1;
```

Fix

```
constant size    : integer := 1;
constant width   : integer := 32
```

15.13.8 constant_010

This rule checks for a single space before the := keyword in constant declarations. Having a space makes it clearer where the assignment occurs on the line.

Violation

```
constant size : integer:= 1;
constant width : integer    := 10;
```

Fix

```
constant size : integer := 1;
constant width : integer := 10;
```

15.13.9 constant_011

This rule checks the constant type has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
constant size : INTEGER := 1;
```

Fix

```
constant size : integer := 1;
```

15.13.10 constant_012

This rule checks the alignment of multiline constants that contain arrays.

Refer to section Configuring Multiline Indent Rules for options.

Note: The structure of multiline array constants is handled by the rule constant_016.

Violation

```
constant rom : romq_type :=
(
         0,
     65535,
     32768
  );
```

Fix

```
constant rom : romq_type :=
(
  0,
  65535,
  32768
);
```

15.13.11 constant_013

This rule checks for consistent capitalization of constant names.

Violation

```
architecture RTL of ENTITY1 is

  constant c_size  : integer := 5;
  constant c_ones  : std_logic_vector(c_size - 1 downto 0) := (others => '1');
  constant c_zeros : std_logic_vector(c_size - 1 downto 0) := (others => '0');

  signal data : std_logic_vector(c_size - 1 downto 0);

begin

  data <= C_ONES;

  PROC_NAME : process () is
  begin

    data <= C_ones;

    if (sig2 = '0') then
      data <= c_Zeros;
    end if;

  end process PROC_NAME;

end architecture RTL;
```

Fix

```
architecture RTL of ENTITY1 is

  constant c_size  : integer := 5;
  constant c_ones  : std_logic_vector(c_size - 1 downto 0) := (others => '1');
  constant c_zeros : std_logic_vector(c_size - 1 downto 0) := (others => '0');

  signal data : std_logic_vector(c_size - 1 downto 0);

begin

  data <= c_ones;

  PROC_NAME : process () is
  begin

    data <= c_ones;

    if (sig2 = '0') then
      data <= c_zeros;
    end if;

  end process PROC_NAME;

end architecture RTL;
```

15.13.12 constant_014

This rule checks the indent of multiline constants that do not contain arrays.

Violation

```
constant width : integer := a + b +
  c + d;
```

Fix

```
constant width : integer := a + b +
                            c + d;
```

15.13.13 constant_015

This rule checks for valid prefixes on constant identifiers. The default constant prefix is *c_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
constant my_const : integer;
```

Fix

```
constant c_my_const : integer;
```

15.13.14 constant_016

This rule checks the structure of multiline constants that contain arrays.

Refer to section Configuring Multiline Structure Rules for options.

Note: The indenting of multiline array constants is handled by the rule constant_012.

Violation

```
constant rom : romq_type := (0, 65535, 32768);
```

Fix

```
constant rom : romq_type :=
(
  0,
  65535,
  32768
);
```

15.13.15 constant_600

This rule checks for valid suffixes on constant identifiers. The default constant suffix is *_c*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
constant my_const : integer;
```

Fix

```
constant my_const_c : integer;
```

15.14 Context Rules

15.14.1 context_001

This rule checks the indent of the **context** keyword.

Violation

```
context c1 is

library ieee;
```

Fix

```
context c1 is

  library ieee;
```

15.14.2 context_002

This rule checks for a single space between the **context** keyword and the context identifier.

Violation

```
context   c1 is
```

Fix

```
context c1 is
```

15.14.3 context_003

This rule checks for blank lines or comments above the **context** keyword.

Refer to Configuring Previous Line Rules for options.

The default style is no_code.

Violation

```
library ieee;
context c1 is

--Some Comment
context c1 is
```

Fix

```
library ieee;

context c1 is

--Some Comment
context c1 is
```

15.14.4 context_004

This rule checks the **context** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
CONTEXT c1 is
```

Fix

```
context c1 is
```

15.14.5 context_005

This rule checks the context identifier is on the same line as the **context** keyword.

Violation

```
context
c1
  is
```

Fix

```
context c1
  is
```

15.14.6 context_006

This rule checks the **is** keyword is on the same line as the context identifier.

Violation

```
context c1
  is
```

Fix

```
context c1 is
```

15.14.7 context_007

This rule checks for code after the **is** keyword.

Violation

```
context c1 is -- Comments are allowed

context c1 is library ieee; -- This is not allowed
```

Fix

```
context c1 is -- Comments are allowed

context c1 is
  library ieee; -- This is not allowed
```

15.14.8 context_008

This rule checks the **end** keyword is on it's own line.

Violation

```
context c1 is library ieee; end context c1;

context c1 is library ieee; end;
```

Fix

```
context c1 is library ieee;
end context c1;

context c1 is library ieee;
end;
```

15.14.9 context_009

This rule checks the **context** keyword is on the same line as the end context keyword.

Violation

```
end
context c1;
```

Fix

```
end context
  c1;
```

15.14.10 context_010

This rule checks the context identifier is on the same line as the end context keyword.

Violation

```
end context
c1;
```

Fix

```
end context c1;
```

15.14.11 context_011

This rule checks the semicolon is on the same line as the **end** keyword.

Violation

```
end
;

end context
;

end context c1
;
```

Fix

```
end;

end context;

end context c1;
```

15.14.12 context_012

This rule checks the context identifier has proper case in the context declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
context C1 is
```

Fix

```
context c1 is
```

15.14.13 context_013

This rule checks the **is** keyword has proper case in the context declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
context c1 IS
```

Fix

```
context c1 is
```

15.14.14 context_014

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
End;

END context;
```

Fix

```
end;

end context;
```

15.14.15 context_015

This rule checks the context keyword has proper case in the end context declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end CONTEXT;
```

Fix

```
end context;
```

15.14.16 context_016

This rule checks the context identifier has proper case in the end context declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end context C1;
```

Fix

```
end context c1;
```

15.14.17 context_017

This rule checks for a single space between the context identifier and the **is** keyword.

Violation

```
context c1     is
```

Fix

```
context c1 is
```

15.14.18 context_018

This rule checks for a single space between the **end** keyword and the **context** keyword.

Violation

```
end;

end    context;
```

Fix

```
end;

end context;
```

15.14.19 context_019

This rule checks for a single space between the **context** keyword and the context identifier.

Violation

```
end context;

end context     c1;
```

Fix

```
end context;

end context c1;
```

15.14.20 context_020

This rule checks the indent of the **end** keyword.

Violation

```
context c1 is
   end context c1;
```

Fix

```
context c1 is
end context c1;
```

15.14.21 context_021

This rule checks for the keyword **context** in the **end context** statement.

Refer to the section Configuring Optional Items for options.

Violation

```
end c1;

end;
```

Fix

```
end context c1;

end context;
```

15.14.22 context_022

This rule checks for the context name in the **end context** statement.

Refer to the section Configuring Optional Items for options.

Violation

```
end context;
```

Fix

```
end context c1;
```

15.14.23 context_023

This rule adds a blank line below the **is** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
context c1 is
  library IEEE;
```

Fix

```
context c1 is

  library IEEE;
```

15.14.24 context_024

This rule checks for blank lines or comments above the **end** keyword.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
  use ieee.std_logic_1164.all;
end context;
```

Fix

```
  use ieee.std_logic_1164.all;

end context;
```

15.14.25 context_025

This rule adds a blank line below the context semicolon.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end context;
entity fifo is
```

Fix

```
end context;

entity fifo is
```

15.14.26 context_026

This rule ensures a single blank line after the **context** keword.

Violation

```
context c1 is



  library ieee;
```

Fix

```
context c1 is

  library ieee;
```

15.14.27 context_027

This rule ensures a single blank line before the **end** keword.

Violation

```
  use ieee.std_logic_1164.all;


```

(continues on next page)

(continued from previous page)

```
end context;
```

Fix

```
  use ieee.std_logic_1164.all;

end context;
```

15.14.28 context_028

Note: This rule has not been implemented yet.

This rule checks for alignment of inline comments in the context declaration.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
context c1 is                       -- Some comment
  library ieee;                        -- Other comment
    use ieee.std_logic_1164.all;   -- Comment 3
end context c1;  -- Comment 4
```

Fix

```
context c1 is                    -- Some comment
  library ieee;                  -- Other comment
    use ieee.std_logic_1164.all; -- Comment 3
end context c1;                  -- Comment 4
```

15.15 Context Reference Rules

15.15.1 context_ref_001

This rule checks the indent of the **context** keyword.

Violation

```
library ieee;
context c1;
```

Fix

```
library ieee;
  context c1;
```

15.15.2 context_ref_002

This rule checks for a single space between the **context** keyword and the context selected name.

Violation

```
context    c1;
```

Fix

```
context c1;
```

15.15.3 context_ref_003

This rule checks the **context** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
CONTEXT c1;
```

Fix

```
context c1;
```

15.15.4 context_ref_004

This rule checks the context selected names have proper case in the context reference.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
context C1;

context CON1, Con2;
```

Fix

```
context c1;

context con1, con2;
```

15.15.5 context_ref_005

This rule checks the **context** keyword is on it's own line.

Violation

```
context c1 is library ieee; context con1; end context c1;

library ieee; context con2;
```

Fix

```
context c1 is library ieee;
context con1; end context c1;

library ieee;
context con2;
```

15.15.6 context_ref_006

This rule checks the semicolon is on the same line as the context selected name.

Note: This rule has not been implemented yet.

Violation

```
context c1
;

context
c1
;
```

Fix

```
context c1;

context
c1;
```

15.15.7 context_ref_007

This rule checks for code after the semicolon.

Note: This rule has not been implemented yet.

Violation

```
context c1; -- Comments are allowed

context c1; library ieee; -- This is not allowed
```

Fix

```
context c1; -- Comments are allowed

context c1;
  library ieee; -- This is not allowed
```

15.15.8 context_ref_008

This rule checks the context selected name is on the same line as the **context** keyword.

Note: This rule has not been implemented yet.

Violation

```
context
c1
;
```

Fix

```
context c1

;
```

15.15.9 context_ref_009

This rule checks for multiple selected names in a single reference.

Note: This rule has not been implemented yet.

Violation

```
context c1, c2, c3; -- Comment 1

context c1,
        c2,
        c3;
```

```
context c1;
context c2;
context c3;

context c1;
context c2;
context c3;
```

15.16 Entity Rules

15.16.1 entity_001

This rule checks the indent of the **entity** keyword.

Violation

```
library ieee;

  entity fifo is
```

Fix

```
library ieee;

entity fifo is
```

15.16.2 entity_002

This rule checks for a single space after the **entity** keyword.

Violation

```
entity    fifo is
```

Fix

```
entity fifo is
```

15.16.3 entity_003

This rule checks for blank lines or comments above the entity keyword.

Refer to the section Configuring Previous Line Rules for options.

Violation

```
library ieee;
entity fifo is
```

Fix

```
library ieee;

entity fifo is
```

15.16.4 entity_004

This rule checks the **entity** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ENTITY fifo is
```

Fix

```
entity fifo is
```

15.16.5 entity_005

This rule checks the **is** keyword is on the same line as the **entity** keyword.

Violation

```
entity fifo

entity fifo
  is
```

Fix

```
entity fifo is

entity fifo is
```

15.16.6 entity_006

This rule checks the **is** keyword has proper case in the entity declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
entity fifo IS
```

Fix

```
entity fifo is
```

15.16.7 entity_007

This rule checks for a single space before the **is** keyword.

Violation

```
entity fifo    is
```

Fix

```
entity fifo is
```

15.16.8 entity_008

This rule checks the entity name has proper case in the entity declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
entity Fifo is
```

Fix

```
entity fifo is
```

15.16.9 entity_009

This rule checks the indent of the **end** keyword.

Violation

```
  wr_en : in    std_logic;
  rd_en : in    std_logic
);
  end entity fifo;
```

Fix

```
    wr_en : in    std_logic;
    rd_en : in    std_logic
  );
end entity fifo;
```

15.16.10 entity_010

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END entity fifo;
```

Fix

```
end entity fifo;
```

15.16.11 entity_011

This rule checks for a single space after the **end** keyword.

Violation

```
end    entity fifo;
```

Fix

```
end entity fifo;
```

15.16.12 entity_012

This rule checks the case of the entity name in the **end entity** statement.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end entity FIFO;
```

Fix

```
end entity fifo;
```

15.16.13 entity_013

This rule checks for a single space after the **entity** keyword in the closing of the entity declaration.

Violation

```
end entity    fifo;
```

Fix

```
end entity fifo;
```

15.16.14 entity_014

This rule checks the **entity** keyword has proper case in the closing of the entity declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end ENTITY fifo;
```

Fix

```
end entity fifo;
```

15.16.15 entity_015

This rule checks for the keyword **entity** in the **end entity** statement.

Refer to the section Configuring Optional Items for options.

Violation

```
end fifo;

end;
```

Fix

```
end entity fifo;

end entity;
```

15.16.16 entity_016

This rule checks for blank lines above the **end entity** keywords.

Violation

```
    wr_en : in    std_logic;
    rd_en : in    std_logic
  );


end entity fifo;
```

Fix

```
    wr_en : in    std_logic;
    rd_en : in    std_logic
  );
end entity fifo;
```

15.16.17 entity_017

This rule checks the alignment of the colon for each generic and port in the entity declaration.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
generic (
    g_width : positive;
    g_output_delay : positive
);
port (
    clk_i : in std_logic;
    data_i : in std_logic;
    data_o : in std_logic
);
```

Fix

```
generic (
    g_width        : positive;
    g_output_delay : positive
);
port (
    clk_i  : in std_logic;
    data_i : in std_logic;
    data_o : in std_logic
);
```

15.16.18 entity_018

This rule checks the alignment of **:=** operator for each generic and port in the entity declaration.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
generic (
    g_width        : positive := 8;
    g_output_delay : positive      := 5
);
port (
    clk_i   : in std_logic;
    data1_i : in std_logic   := 'X';
    data2_i : in std_logic       := 'X';
    data_o  : in std_logic
);
```

Fix

```
generic (
    g_width        : positive := 8;
    g_output_delay : positive := 5
);
port (
    clk_i   : in std_logic;
    data1_i : in std_logic := 'X';
    data2_i : in std_logic := 'X';
```

(continues on next page)

(continued from previous page)

```
    data_o   : in std_logic
);
```

15.16.19 entity_019

This rule checks for the entity name in the **end entity** statement.

Refer to the section Configuring Optional Items for options.

Violation

```
end entity;
```

Fix

```
end entity entity_name;
```

15.16.20 entity_020

This rule checks for alignment of inline comments in the entity declaration.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
generic (
    g_width        : positive;   -- Data width
    g_output_delay : positive -- Delay at output
);
port (
    clk_i   : in std_logic; -- Input clock
    data_i  : in std_logic;    -- Data input
    data_o  : in std_logic -- Data output
);
```

Fix

```
generic (
    g_width        : positive; -- Data width
    g_output_delay : positive  -- Delay at output
);
port (
    clk_i   : in std_logic; -- Input clock
    data_i  : in std_logic; -- Data input
    data_o  : in std_logic  -- Data output
);
```

15.16.21 entity_600

This rule checks for consistent capitalization of generic names in entity declarations.

Violation

```
entity FIFO is
  generic (
    G_WIDTH : natural := 16
  );
  port (
    I_DATA : std_logic_vector(g_width - 1 downto 0);
    O_DATA : std_logic_vector(g_width - 1 downto 0)
  );
end entity fifo;
```

Fix

```
entity FIFO is
  generic (
    G_WIDTH : natural := 16
  );
  port (
    I_DATA : std_logic_vector(G_WIDTH - 1 downto 0);
    O_DATA : std_logic_vector(G_WIDTH - 1 downto 0)
  );
end entity fifo;
```

15.17 Entity Specification Rules

15.17.1 entity_specification_100

This rule checks for a single space after the colon.

Violation

```
attribute coordinate of comp_1 :component is (0.0, 17.5);
attribute coordinate of comp_1 :    component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.17.2 entity_specification_101

This rule checks for at least a single space before the colon.

Violation

```
attribute coordinate of comp_1: component is (0.0, 17.5);

attribute coordinate of comp_1      : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);

attribute coordinate of comp_1      : component is (0.0, 17.5);
```

15.17.3 entity_specification_500

This rule checks the **others** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate of OTHERS : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of others : component is (0.0, 17.5);
```

15.17.4 entity_specification_501

This rule checks the **all** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate of ALL : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of all : component is (0.0, 17.5);
```

15.17.5 entity_specification_502

This rule checks the *entity_designator* has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate of COMP_1, COMP_2 : component is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1, comp_2 : component is (0.0, 17.5);
```

15.17.6 entity_specification_503

This rule checks the *entity_class* has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
attribute coordinate of comp_1 : COMPONENT is (0.0, 17.5);
```

Fix

```
attribute coordinate of comp_1 : component is (0.0, 17.5);
```

15.18 Exit Rules

15.18.1 exit_statement_300

This rule checks the indent of the **exit** keyword.

Violation

```
end if;

  exit;
```

Fix

```
end if;

exit;
```

15.19 File Rules

15.19.1 file_001

This rule checks the indent of **file** declarations.

Violation

```
architecture rtl of fifo is

file defaultImage : load_file_type open read_mode is load_file_name;

```

(continues on next page)

(continued from previous page)

```
file defaultImage : load_file_type open read_mode
is load_file_name;

begin
```

Fix

```
architecture rtl of fifo is

  file defaultImage : load_file_type open read_mode is load_file_name;

  file defaultImage : load_file_type open read_mode
    is load_file_name;

begin
```

15.19.2 file_002

This rule checks the **file** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
architecture rtl of fifo is

  FILE defaultImage : load_file_type open read_mode is load_file_name;

begin
```

Fix

```
architecture rtl of fifo is

  file defaultImage : load_file_type open read_mode is load_file_name;

begin
```

15.19.3 file_003

This rule was depricated and replaced with rules:

- function_015
- package_019
- procedure_010
- architecture_029

15.20 For Loop Rules

15.20.1 for_loop_001

This rule checks the indentation of the **for** keyword.

Violation

```
fifo_proc : process () is
begin

for index in 4 to 23 loop

  end loop;

end process;
```

Fix

```
fifo_proc : process () is
begin

  for index in 4 to 23 loop

  end loop;

end process;
```

15.20.2 for_loop_002

This rule checks the indentation of the **end loop** keywords.

Violation

```
fifo_proc : process () is
begin

  for index in 4 to 23 loop

      end loop;

end process;
```

Fix

```
fifo_proc : process () is
begin

  for index in 4 to 23 loop

  end loop;

end process;
```

15.20.3 for_loop_003

This rule checks the proper case of the label on a foor loop.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
LABEL : for index in 4 to 23 loop
Label : for index in 0 to 100 loop
```

Fix

```
label : for index in 4 to 23 loop
label : for index in 0 to 100 loop
```

15.20.4 for_loop_004

This rule checks if a label exists on a for loop that a single space exists between the label and the colon.

Violation

```
label: for index in 4 to 23 loop
label     : for index in 0 to 100 loop
```

Fix

```
label : for index in 4 to 23 loop
label : for index in 0 to 100 loop
```

15.20.5 for_loop_005

This rule checks if a label exists on a for loop that a single space exists after the colon.

Violation

```
label :    for index in 4 to 23 loop
label :  for index in 0 to 100 loop
```

Fix

```
label : for index in 4 to 23 loop
label : for index in 0 to 100 loop
```

15.21 Function Rules

15.21.1 function_001

This rule checks the indentation of the **function** keyword.

Violation

```
architecture RTL of FIFO is

    function overflow (a: integer) return integer is


function underflow (a: integer) return integer is

begin
```

Fix

```
architecture RTL of FIFO is

  function overflow (a: integer) return integer is

  function underflow (a: integer) return integer is

begin
```

15.21.2 function_002

This rule checks a single space exists after the **function** keyword.

Violation

```
function     overflow (a: integer) return integer is
```

Fix

```
function overflow (a: integer) return integer is
```

15.21.3 function_003

This rule checks for a single space between the function name and the (.'

Violation

```
function overflow   (a: integer) return integer is

function underflow(a: integer) return integer is
```

Fix

```
function overflow (a: integer) return integer is

function underflow (a: integer) return integer is
```

15.21.4 function_004

This rule checks the **begin** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
function overflow (a: integer) return integer is
BEGIN
```

Fix

```
function overflow (a: integer) return integer is
begin
```

15.21.5 function_005

This rule checks the **function** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
FUNCTION overflow (a: integer) return integer is
```

Fix

```
function overflow (a: integer) return integer is
```

15.21.6 function_006

This rule checks for blank lines or comments above the **function** keyword.

Refer to Configuring Previous Line Rules for options.

Violation

```
architecture RTL of FIFO is
  function overflow (a: integer) return integer is
```

Fix

```
architecture RTL of FIFO is

  function overflow (a: integer) return integer is
```

15.21.7 function_007

This rule checks for a blank line below the end of the function declaration.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
function overflow (a: integer) return integer is
end;
signal wr_en : std_logic;
```

Fix

```
function overflow (a: integer) return integer is
end;

signal wr_en : std_logic;
```

15.21.8 function_008

This rule checks the indent of function parameters on multiple lines.

Violation

```
function func_1 (a : integer; b : integer;
            c : unsigned(3 downto 0);
    d : std_logic_vector(7 downto 0);
       e : std_logic) return integer is
begin

end;
```

Fix

```
function func_1 (a : integer; b : integer;
  c : unsigned(3 downto 0);
  d : std_logic_vector(7 downto 0);
  e : std_logic) return integer is
begin

end;
```

15.21.9 function_009

This rule checks for a function parameter on the same line as the function keyword when the parameters are on multiple lines.

Violation

```
function func_1 (a : integer; b : integer;
  c : unsigned(3 downto 0);
  d : std_logic_vector(7 downto 0);
  e : std_logic) return integer is
begin

end;
```

Fix

```
function func_1 (
  a : integer; b : integer;
  c : unsigned(3 downto 0);
  d : std_logic_vector(7 downto 0);
  e : std_logic) return integer is
begin

end;
```

15.21.10 function_010

This rule checks for consistent capitalization of function names.

Violation

```
architecture rtl of fifo is

  function func_1 ()

begin

  OUT1 <= Func_1;

  PROC1 : process () is
  begin

     sig1 <= FUNC_1;

  end process;

end architecture rtl;
```

Fix

```
architecture rtl of fifo is

  function func_1 ()

begin

  OUT1 <= func_1;

  PROC1 : process () is
  begin
```

(continues on next page)

(continued from previous page)

```
    sig1 <= func_1;

  end process;

end architecture rtl;
```

15.21.11 function_012

This rule checks the colons are in the same column for all declarations in the function declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
variable var1 : natural;
variable var2  : natural;
constant c_period : time;
```

Fix

```
variable var1     : natural;
variable var2     : natural;
constant c_period : time;
```

15.21.12 function_013

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END;

End function foo;
```

Fix

```
end;

end function foo;
```

15.21.13 function_014

This rule checks the **function** keyword in the **end function** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end FUNCTION;

end Function foo;
```

Fix

```
end function;

end function foo;
```

15.21.14 function_015

This rule checks the identifiers for all declarations are aligned in the function declarative part.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable var1 : natural;
signal sig1 : natural;
constant c_period : time;
```

Fix

```
variable var1     : natural;
signal   sig1     : natural;
constant c_period : time;
```

15.21.15 function_016

This rule checks the indent of return statements in function bodies.

Violation

```
function func1 return integer is
begin
     return 99;
return 99;
end func1;
```

Fix

```
function func1 return integer is
begin
  return 99;
  return 99;
end func1;
```

15.21.16 function_017

This rule checks the function designator has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
function OVERflow (a: integer) return integer is
```

Fix

```
function overflow (a: integer) return integer is
```

15.22 Generate Rules

15.22.1 generate_001

This rule checks the indent of the generate declaration.

Violation

```
architecture rtl of fifo is
begin

ram_array : for i in 0 to 7 generate

      ram_array : for i in 0 to 7 generate
```

Fix

```
architecture rtl of fifo is
begin

  ram_array : for i in 0 to 7 generate

  ram_array : for i in 0 to 7 generate
```

15.22.2 generate_002

This rule checks for a single space between the label and the colon.

Violation

```
ram_array: for i in 0 to 7 generate
```

Fix

```
ram_array : for i in 0 to 7 generate
```

15.22.3 generate_003

This rule checks for a blank line below the **end generate** keywords.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end generate ram_array;
wr_en <= '1';
```

Fix

```
end generate ram_array;

wr_en <= '1';
```

15.22.4 generate_004

This rule checks for blank lines or comments before the **generate** label.

Refer to Configuring Previous Line Rules for options.

Violation

```
wr_en <= '1';
ram_array : for i in 0 to 7 generate
```

Fix

```
wr_en <= '1';

ram_array : for i in 0 to 7 generate
```

15.22.5 generate_005

This rule checks the generate label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
RAM_ARRAY: for i in 0 to 7 generate
```

Fix

```
ram_array: for i in 0 to 7 generate
```

15.22.6 generate_006

This rule checks the indent of the **begin** keyword.

Violation

```
ram_array : for i in 0 to 7 generate
   begin
```

Fix

```
ram_array : for i in 0 to 7 generate
begin
```

15.22.7 generate_007

This rule checks the indent of the **end generate** keyword.

Violation

```
ram_array : for i in 0 to 7 generate
begin
  end generate ram_array;
```

Fix

```
ram_array : for i in 0 to 7 generate
begin
end generate ram_array;
```

15.22.8 generate_008

This rule checks for a single space after the **end** keyword.

Violation

```
end   generate ram_array;
```

Fix

```
end generate ram_array;
```

15.22.9 generate_009

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END generate ram_array;
```

Fix

```
end generate ram_array;
```

15.22.10 generate_010

This rule checks the **generate** keyword has the proper case in the **end generate** line.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end GENERATE ram_array;
```

Fix

```
end generate ram_array;
```

15.22.11 generate_011

This rule checks the **end generate** line has a label on for generate statements.

Violation

```
ram_array : for i in 0 to 127 generate

end generate;
```

Fix

```
ram_array : for i in 0 to 127 generate

end generate ram_array;
```

15.22.12 generate_012

This rule checks the **end generate** label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end generate RAM_ARRAY;
```

Fix

```
end generate ram_array;
```

15.22.13 generate_013

This rule checks for a single space after the **generate** keyword and the label in the **end generate** keywords.

Violation

```
end generate    ram_array;
```

Fix

```
end generate ram_array;
```

15.22.14 generate_014

This rule checks for a single space between the colon and the **for** keyword.

Violation

```
ram_array :for i in 0 to 7 generate
ram_array :   for i in 0 to 7 generate
```

Fix

```
ram_array : for i in 0 to 7 generate
ram_array : for i in 0 to 7 generate
```

15.22.15 generate_015

This rule checks the generate label and the **generate** keyword are on the same line. Keeping the label and generate on the same line reduces excessive indenting.

Violation

```
ram_array :
  for i in 0 to 7 generate
```

Fix

```
ram_array : for i in 0 to 7 generate
```

15.22.16 generate_016

This rule checks the indent of the **when** keyword in generate case statements.

Violation

```
GEN_LABEL : case condition generate
  when 0 =>
    when 1 =>
when 2 =>
```

Fix

```
GEN_LABEL : case condition generate
  when 0 =>
  when 1 =>
  when 2 =>
```

15.22.17 generate_017

This rule checks for valid prefixes on generate statement labels. The default prefix is *gen_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
label : case condition generate
```

Fix

```
gen_label : case condition generate
```

15.22.18 generate_018

This rule checks the indent of the **end** keyword in the generate statement body.

Violation

```
ram_array : for i in 0 to 7 generate
begin
  end;
end generate;
```

Fix

```
ram_array : for i in 0 to 7 generate
begin
end;
end generate;
```

15.22.19 generate_400

This rule checks the identifiers for all declarations are aligned in the generate declarative part in for generate statements.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable    var1 : natural;
constant  c_period : time;
```

Fix

```
variable var1     : natural;
constant c_period : time;
```

15.22.20 generate_401

This rule checks the colons are in the same column for all declarations in the generate declarative part in for generate statements.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
signal sig1: natural;
variable var2  : natural;
constant c_period : time;
file my_test_input : my_file_type;
```

Fix

```
signal sig1         : natural;
variable var2       : natural;
constant c_period   : time;
file my_test_input : my_file_type;
```

15.22.21 generate_402

This rule checks the identifiers for all declarations are aligned in the generate declarative part in if generate statements.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable    var1 : natural;
constant  c_period : time;
```

Fix

```
variable var1     : natural;
constant c_period : time;
```

15.22.22 generate_403

This rule checks the colons are in the same column for all declarations in the generate declarative part in if generate statements.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
signal sig1: natural;
variable var2  : natural;
constant c_period : time;
file my_test_input : my_file_type;
```

Fix

```
signal sig1          : natural;
variable var2        : natural;
constant c_period    : time;
file my_test_input : my_file_type;
```

15.22.23 generate_404

This rule checks the identifiers for all declarations are aligned in the generate declarative part in case generate statements.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable    var1 : natural;
constant  c_period : time;
```

Fix

```
variable var1     : natural;
constant c_period : time;
```

15.22.24 generate_405

This rule checks the colons are in the same column for all declarations in the generate declarative part in case generate statements.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
signal sig1: natural;
variable var2  : natural;
constant c_period : time;
file my_test_input : my_file_type;
```

Fix

```
signal sig1          : natural;
variable var2        : natural;
constant c_period    : time;
file my_test_input : my_file_type;
```

15.22.25 generate_600

This rule checks for valid suffixes on generate statement labels. The default suffix is *_gen*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
label : case condition generate
```

Fix

```
label_gen : case condition generate
```

15.23 Generic Rules

15.23.1 generic_001

This rule checks for blank lines above the **generic** keyword.

Violation

```
entity fifo is



  generic (
```

Fix

```
entity fifo is
  generic (
```

15.23.2 generic_002

This rule checks the indent of the **generic** keyword.

Violation

```
entity fifo is
     generic (

entity fifo is
generic (
```

Fix

```
entity fifo is
  generic (

entity fifo is
  generic (
```

15.23.3 generic_003

This rule checks for a single space between the **generic** keyword and the (.

Violation

```
generic     (

generic(
```

Fix

```
generic (

generic (
```

15.23.4 generic_004

This rule checks the indent of generic declarations.

Violation

```
generic (
g_width : integer := 32;
       g_depth : integer := 512
)
```

Fix

```
generic (
  g_width : integer := 32;
  g_depth : integer := 512
)
```

15.23.5 generic_005

This rule checks for a single space after the colon in a generic declaration.

Violation

```
g_width :integer := 32;
```

Fix

```
g_width : integer := 32;
```

15.23.6 generic_006

This rule checks for a single space after the default assignment.

Violation

```
g_width : integer :=32;
g_depth : integer :=     512;
```

Fix

```
g_width : integer := 32;
g_depth : integer := 512;
```

15.23.7 generic_007

This rule checks the generic names have proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
G_WIDTH : integer := 32;
```

Fix

```
g_width : integer := 32;
```

15.23.8 generic_008

This rule checks the indent of the closing parenthesis.

Violation

```
g_depth : integer := 512
);
```

Fix

```
  g_depth : integer := 512
);
```

15.23.9 generic_009

This rule checks the **generic** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
GENERIC (
```

Fix

```
generic (
```

15.23.10 generic_010

This rule checks the closing parenthesis is on a line by itself.

Violation

```
g_depth : integer := 512);
```

Fix

```
  g_depth : integer := 512
);
```

15.23.11 generic_013

This rule checks for the **generic** keyword on the same line as a generic declaration.

Violation

```
generic (g_depth : integer := 512;
```

Fix

```
generic (
  g_depth : integer := 512;
```

15.23.12 generic_014

This rule checks for at least a single space before the colon.

Violation

```
g_address_width: integer := 10;
g_data_width : integer := 32;
g_depth: integer := 512;
```

Fix

```
g_address_width : integer := 10;
g_data_width : integer := 32;
g_depth : integer := 512;
```

15.23.13 generic_016

This rule checks for multiple generics defined on a single line.

Violation

```
generic (
  g_width : std_logic := '0';g_depth : std_logic := '1'
);
```

Fix

```
generic (
  g_width : std_logic := '0';
  g_depth : std_logic := '1'
);
```

15.23.14 generic_017

This rule checks the generic type has proper case if it is a VHDL keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
generic (
  g_width : STD_LOGIC := '0';
  g_depth : Std_logic := '1'
);
```

Fix

```
generic (
  g_width : std_logic := '0';
  g_depth : std_logic := '1'
);
```

15.23.15 generic_018

This rule checks the **generic** keyword is on the same line as the (.

Violation

```
generic
 (
```

Fix

```
generic (
```

15.23.16 generic_019

This rule checks for blank lines before the); of the generic declaration.

Violation

```
generic (
  g_width : std_logic := '0';
  g_depth : Std_logic := '1'


);
```

Fix

```
generic (
  g_width : std_logic := '0';
  g_depth : Std_logic := '1'
);
```

15.23.17 generic_020

This rule checks for valid prefixes on generic identifiers. The default generic prefix is *g_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
generic(my_generic : integer);
```

Fix

```
generic(g_my_generic : integer);
```

15.23.18 generic_600

This rule checks for valid suffixes on generic identifiers. The default generic suffix is *_g*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
generic(my_generic : integer);
```

Fix

```
generic(my_generic_g : integer);
```

15.24 Generic Map Rules

15.24.1 generic_map_001

This rule checks the **generic map** keywords have proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
GENERIC MAP (
```

Fix

```
generic map (
```

15.24.2 generic_map_002

This rule checks generic names have proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
generic map (
  DEPTH => 512,
  WIDTH => 32
)
```

Fix

```
generic map (
  depth => 512,
  width => 32
)
```

15.24.3 generic_map_003

This rule checks the (is on the same line as the **generic map** keywords.

Violation

```
generic map
(
  WIDTH => 32,
  DEPTH => 512
)
```

Fix

Use explicit port mapping.

```
generic map (
  WIDTH => 32,
  DEPTH => 512
)
```

15.24.4 generic_map_004

This rule checks for the closing parenthesis *)* on generic maps are on their own line.

Violation

```
generic map (
  GENERIC_1 => 0,
  GENERIC_2 => TRUE,
  GENERIC_3 => FALSE)
```

Fix

```
generic map (
  GENERIC_1 => 0,
  GENERIC_2 => TRUE,
  GENERIC_3 => FALSE
)
```

15.24.5 generic_map_005

This rule checks if the **generic map** keywords and a generic assignment are on the same line.

Violation

```
generic map (DEPTH => 512,
  WIDTH => 32
)
```

Fix

```
generic map (
  DEPTH => 512,
  WIDTH => 32
)
```

15.24.6 generic_map_006

This rule checks for a single space between the **map** keyword and the (.

Violation

```
generic map(

generic map    (
```

Fix

```
generic map (

generic map (
```

15.24.7 generic_map_007

This rule checks for a single space after the **=>** keyword in generic maps.

Violation

```
generic map
(
  WIDTH =>     32,
  DEPTH => 512
)
```

Fix

```
generic map
(
  WIDTH => 32,
  DEPTH => 512
)
```

15.24.8 generic_map_008

This rule checks for positional generics. Positional ports and generics are subject to problems when the position of the underlying component changes.

Violation

```
port map (
  WR_EN, RD_EN, OVERFLOW
);
```

Fix

Use explicit port mapping.

```
port map (
  WR_EN    => WR_EN,
  RD_EN    => RD_EN,
  OVERFLOW => OVERFLOW
);
```

15.25 If Rules

15.25.1 if_001

This rule checks the indent of the **if** keyword.

Violation

```
  if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

Fix

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

15.25.2 if_002

This rule checks the bolean expression is enclosed in ().

Note: There is a configuration option **parenthesis** which will either insert or remove the parenthesis.

parenthesis set to 'insert' (Default)

Violation

```
if a = '1' then
```

Fix

```
if (a = '1') then
```

parenthesis set to 'remove'

Violation

```
if (a = '1') then
```

Fix

```
if a = '1' then
```

15.25.3 if_003

This rule checks for a single space between the **if** keyword and the (.

Violation

```
if(a = '1') then

if   (a = '1') then
```

Fix

```
if (a = '1') then

if (a = '1') then
```

15.25.4 if_004

This rule checks for a single space between the) and the **then** keyword.

Violation

```
if (a = '1')then

if (a = '1')    then
```

Fix

```
if (a = '1') then

if (a = '1') then
```

15.25.5 if_005

This rule checks for a single space between the **elsif** keyword and the (.

Violation

```
elsif(c = '1') then

elsif   (c = '1') then
```

Fix

```
elsif (c = '1') then

elsif (c = '1') then
```

15.25.6 if_006

This rule checks for empty lines after the **then** keyword.

Violation

```
if (a = '1') then


  b <= '0'
```

Fix

```
if (a = '1') then
  b <= '0'
```

15.25.7 if_007

This rule checks for empty lines before the **elsif** keyword.

Violation

```
  b <= '0'



elsif (c = '1') then
```

Fix

```
  b <= '0'
elsif (c = '1') then
```

15.25.8 if_008

This rule checks for empty lines before the **end if** keywords.

Violation

```
  e <= '0';


end if;
```

Fix

```
  e <= '0';
end if;
```

15.25.9 if_009

This rule checks the alignment of multiline boolean expressions.

Violation

```
if (a = '0' and b = '1' and
      c = '0') then
```

Fix

```
if (a = '0' and b = '1' and
    c = '0') then
```

15.25.10 if_010

This rule checks for empty lines before the **else** keyword.

Violation

```
  d <= '1';


else
```

Fix

```
  d <= '1';
else
```

15.25.11 if_011

This rule checks for empty lines after the **else** keyword.

Violation

```
else


  e <= '0';
```

Fix

```
else
  e <= '0';
```

15.25.12 if_012

This rule checks the indent of the **elsif** keyword.

Violation

```
if (a = '1') then
  b <= '0'
  elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

Fix

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

15.25.13 if_013

This rule checks the indent of the **else** keyword.

Violation

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
  else
  e <= '0';
end if;
```

Fix

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

15.25.14 if_014

This rule checks the indent of the **end if** keyword.

Violation

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
  end if;
```

Fix

```
if (a = '1') then
  b <= '0'
elsif (c = '1') then
  d <= '1';
else
  e <= '0';
end if;
```

15.25.15 if_015

This rule checks for a single space between the **end if** keywords.

Violation

```
end     if;
```

Fix

```
end if;
```

15.25.16 if_020

This rule checks the **end if** keyword is on it's own line.

Violation

```
if (a = '1') then c <= '1'; else c <= '0'; end if;
```

Fix

```
if (a = '1') then c <= '1'; else c <= '0';
end if;
```

15.25.17 if_021

This rule checks the **else** keyword is on it's own line.

Violation

```
if (a = '1') then c <= '1'; else c <= '0'; end if;
```

Fix

```
if (a = '1') then c <= '0';
else c <= '1'; end if;
```

15.25.18 if_022

This rule checks for code after the **else** keyword.

Violation

```
if (a = '1') then c <= '1'; else c <= '0'; end if;
```

Fix

```
if (a = '1') then c <= '1'; else
  c <= '0'; end if;
```

15.25.19 if_023

This rule checks the **elsif** keyword is on it's own line.

Violation

```
if (a = '1') then c <= '1'; else c <= '0'; elsif (b = '0') then d <= '0'; end if;
```

Fix

```
if (a = '1') then c <= '1'; else c <= '0';
elsif (b = '0') then d <= '0'; end if;
```

15.25.20 if_024

This rule checks for code after the **then** keyword.

Violation

```
if (a = '1') then c <= '1';
```

Fix

```
if (a = '1') then
  c <= '1';
```

15.25.21 if_025

This rule checks the **if** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
IF (a = '1') then
```

Fix

```
if (a = '1') then
```

15.25.22 if_026

This rule checks the **elsif** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ELSIF (a = '1') then
```

Fix

```
elsif (a = '1') then
```

15.25.23 if_027

This rule checks the **else** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
ELSE
```

Fix

```
else
```

15.25.24 if_028

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END if;

End if;
```

Fix

```
end if;

end if;
```

15.25.25 if_029

This rule checks the **then** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
if (a = '1') THEN
```

Fix

```
if (a = '1') then
```

15.25.26 if_030

This rule checks a single blank line after the **end if**. In the case of nested **if** statements, the rule will be enfoced on the last **end if**.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
if (A = '1') then
  B <= '0';
end if;
C <= '1';
```

Fix

```
if (A = '1') then
  B <= '0';
end if;

C <= '1';
```

15.25.27 if_031

This rule checks for blank lines or comments above the **if** keyword. In the case of nested **if** statements, the rule will be enfoced on the first **if**.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
C <= '1';
if (A = '1') then
  B <= '0';
end if;

-- This is a comment
if (A = '1') then
  B <= '0';
end if;
```

Fix

```
C <= '1';

if (A = '1') then
  B <= '0';
end if;

-- This is a comment
if (A = '1') then
  B <= '0';
end if;
```

15.25.28 if_032

This rule aligns consecutive comment only lines above the **elsif** keyword in if statements. These comments are used to describe what the elsif code is going to do.

Violation

```
      -- comment 1
-- comment 2
   -- comment 3
  elsif (a = '1')
    rd_en <= '0';
```

Fix

```
-- comment 1
-- comment 2
-- comment 3
elsif (a = '1')
  rd_en <= '0';
```

15.25.29 if_033

This rule aligns consecutive comment only lines above the **else** keyword in if statements. These comments are used to describe what the elsif code is going to do.

Violation

```
      -- comment 1
-- comment 2
   -- comment 3
  else
    rd_en <= '0';
```

Fix

```
-- comment 1
-- comment 2
-- comment 3
else
  rd_en <= '0';
```

15.25.30 if_034

This rule checks the **if** keyword in the **end if** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end If;

end IF;
```

Fix

```
end if;

end if;
```

15.25.31 if_035

This rule checks the expression after the **if** or **elsif** keyword starts on the same line.

Violation

```
if
  a = '1' then

elsif
  b = '1' then
```

Fix

```
if a = '1' then

elsif b = '1' then
```

15.26 Instantiation Rules

15.26.1 instantiation_001

This rule checks for the proper indentation of instantiations.

Violation

```
   U_FIFO : FIFO
port map (
         WR_EN    => wr_en,
 RD_EN    => rd_en,
       OVERFLOW => overflow
              );
```

Fix

```
U_FIFO : FIFO
  port map (
    WR_EN    => wr_en,
    RD_EN    => rd_en,
    OVERFLOW => overflow
  );
```

15.26.2 instantiation_002

This rule checks for a single space after the colon.

Violation

```
U_FIFO :FIFO
```

Fix

```
U_FIFO : FIFO
```

15.26.3 instantiation_003

This rule checks for a single space before the colon.

Violation

```
U_FIFO: FIFO
```

Fix

```
U_FIFO : FIFO
```

15.26.4 instantiation_004

This rule checks for blank lines or comments above the instantiation.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
WR_EN <= '1';
U_FIFO : FIFO

-- Instantiate another FIFO
U_FIFO2 : FIFO
```

Fix

```
WR_EN <= '1';

U_FIFO : FIFO

-- Instantiate another FIFO
U_FIFO2 : FIFO
```

15.26.5 instantiation_005

This rule checks the **port map** keywords are on their own line.

Violation

```
U_FIFO : FIFO port map (
```

Fix

```
U_FIFO : FIFO
  port map (
```

15.26.6 instantiation_006

This rule has been renamed to port_map_001.

15.26.7 instantiation_007

This rule has been renamed to port_map_004.

15.26.8 instantiation_008

This rule checks the instance label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
U_FIFO : fifo
```

Fix

```
u_fifo : fifo
```

15.26.9 instantiation_009

This rule checks the component name has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
u_fifo : FIFO
```

Fix

```
u_fifo : fifo
```

15.26.10 instantiation_010

This rule checks the alignment of the **=>** operator for each generic and port in the instantiation.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
U_FIFO : FIFO
  generic map (
    g_width => 8,
    g_delay     => 2
  )
  port map (
    wr_en => wr_en,
    rd_en => rd_en,
    overflow => overflow
  );
```

Fix

```
U_FIFO : FIFO
  generic map (
    g_width => 8,
    g_delay => 2
  )
  port map (
    wr_en    => wr_en,
    rd_en    => rd_en,
    overflow => overflow
  );
```

15.26.11 instantiation_011

This rule has been renamed to port_map_002.

15.26.12 instantiation_012

This rule checks the instantiation declaration and the **generic map** keywords are not on the same line.

Violation

```
U_FIFO : FIFO generic map (
```

Fix

```
U_FIFO : FIFO
  generic map (
```

15.26.13 instantiation_013

This rule has been renamed to generic_map_001.

15.26.14 instantiation_014

This rule has been renamed to generic_map_004.

15.26.15 instantiation_016

This rule has been renamed to generic_map_002.

15.26.16 instantiation_017

This rule has been renamed to generic_map_005.

15.26.17 instantiation_018

This rule has been renamed to generic_map_006.

15.26.18 instantiation_019

This rule checks for a blank line below the end of the instantiation declaration.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
U_FIFO : FIFO
  port map (
    WR_EN    => wr_en,
    RD_EN    => rd_en,
    OVERFLOW => overflow
  );
U_RAM : RAM
```

Fix

```
U_FIFO : FIFO
  port map (
    WR_EN    => wr_en,
    RD_EN    => rd_en,
    OVERFLOW => overflow
  );

U_RAM : RAM
```

15.26.19 instantiation_020

This rule has been renamed to port_map_005.

15.26.20 instantiation_021

This rule has been renamed to port_map_009.

15.26.21 instantiation_022

This rule has been renamed to port_map_007.

15.26.22 instantiation_023

This rule checks for comments at the end of the port and generic assignments in instantiations. These comments represent additional maintainence. They will be out of sync with the entity at some point. Refer to the entity for port types, port directions and purpose.

Violation

```
WR_EN => w_wr_en;    -- out : std_logic
RD_EN => w_rd_en;    -- Reads data when asserted
```

Fix

```
WR_EN => w_wr_en;
RD_EN => w_rd_en;
```

15.26.23 instantiation_024

This rule has been split into:

- generic_map_008
- port_map_008

15.26.24 instantiation_025

This rule has been renamed to port_map_003.

15.26.25 instantiation_026

This rule has been renamed to generic_map_003.

15.26.26 instantiation_027

This rule checks the **entity** keyword has proper case in direct instantiations.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
INSTANCE_NAME : ENTITY library.ENTITY_NAME
```

Fix

```
INSTANCE_NAME : entity library.ENTITY_NAME
```

15.26.27 instantiation_028

This rule checks the entity name has proper case in direct instantiations.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
instance_name : entity library.ENTITY_NAME
```

Fix

```
instance_name : entity library.entity_name
```

15.26.28 instantiation_029

This rule checks for alignment of inline comments in an instantiation.

Following extra configurations are supported:

- `separate_generic_port_alignment`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations. **Violation**

Violation

```
wr_en    => write_enable,        -- Wrte enable
rd_en    => read_enable,    -- Read enable
overflow => overflow,         -- FIFO has overflowed
```

Fix

```
wr_en    => write_enable, -- Wrte enable
rd_en    => read_enable,  -- Read enable
overflow => overflow,     -- FIFO has overflowed
```

15.26.29 instantiation_030

This rule has been renamed to generic_map_007.

15.26.30 instantiation_031

This rule checks the component keyword has proper case in component instantiations that use the **component** keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
instance_name : COMPONENT entity_name
```

Fix

```
instance_name : component entity_name
```

15.26.31 instantiation_032

This rule checks for a single space after the **component** keyword if it is used.

Violation

```
INSTANCE_NAME : component ENTITY_NAME
INSTANCE_NAME : component    ENTITY_NAME
INSTANCE_NAME : component   ENTITY_NAME
```

Fix

```
INSTANCE_NAME : component ENTITY_NAME
INSTANCE_NAME : component ENTITY_NAME
INSTANCE_NAME : component ENTITY_NAME
```

15.26.32 instantiation_033

This rule checks for the **component** keyword for a component instantiation.

Refer to the section Configuring Optional Items for options.

Violation

```
INSTANCE_NAME : ENTITY_NAME
```

Fix

```
INSTANCE_NAME : component ENTITY_NAME
```

15.26.33 instantiation_034

This rule checks for component versus direct instantiations.

Refer to the section Configuring Type of Instantiation for options to configure the allowed configuration.

component instantiation

Note: This is the default configuration

Violation

```
U_FIFO : entity fifo_dsn.FIFO(RTL)
```

entity instantiation

Violation

```
U_FIFO : component FIFO

U_FIFO : FIFO
```

15.26.34 instantiation_600

This rule checks for valid suffixes on instantiation labels. The default suffix is *_inst*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
fifo_32x2k : FIFO
```

Fix

```
fifo_32x2k_inst : FIFO
```

15.26.35 instantiation_601

This rule checks for valid prefixes on instantiation labels. The default prefix is *inst_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
fifo_32x2k : FIFO
```

Fix

```
inst_fifo_32x2k : FIFO
```

15.27 Length Rules

These rules cover the length of lines in the VHDL file.

15.27.1 length_001

This rule checks the length of the line.

Refer to the section Configuring Length Rules for configuring this option.

15.27.2 length_002

This rule checks the length of a file.

Refer to the section Configuring Length Rules for configuring this option.

15.27.3 length_003

This rule checks the length of a process statement.

Refer to the section Configuring Length Rules for configuring this option.

15.28 Library Rules

15.28.1 library_001

This rule checks the indent of the **library** keyword. Indenting helps in comprehending the code.

Violation

```
library ieee;
   library fifo_dsn;
```

Fix

```
library ieee;
library fifo_dsn;
```

15.28.2 library_002

This rule checks for excessive spaces after the **library** keyword.

Violation

```
library    ieee;
```

Fix

```
library ieee;
```

15.28.3 library_003

This rule checks for blank lines or comments above the entity keyword.

Refer to the section Configuring Previous Line Rules for options.

There is an additional **allow_library_clause** option which can be set. Refer to section Reporting Single Rule Configuration for details on finding configuration options for individual rules.

allow_library_clause

When set to **True**, it allows consecutive library clauses.

Violation

```
library ieee;
  use ieee.std_logic_1164.all;
library top_dsn;
library fifo_dsn;
```

Fix

```
library ieee;
  use ieee.std_logic_1164.all;

library top_dsn;
library fifo_dsn;
```

15.28.4 library_004

This rule checks the **library** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
Library ieee;

LIBRARY fifo_dsn;
```

Fix

```
library ieee;

library fifo_dsn;
```

15.28.5 library_005

This rule checks the **use** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
library ieee;
  USE ieee.std_logic_1164.all;
  Use ieee.std_logic_unsigned.all;
```

Fix

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;
```

15.28.6 library_006

This rule checks for excessive spaces after the **use** keyword.

Violation

```
library ieee;
  use     ieee.std_logic_1164.all;
  use   ieee.std_logic_unsigned.all;
```

Fix

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;
```

15.28.7 library_007

This rule checks for blank lines or comments above the **process** declaration.

Refer to the section Configuring Blank Lines for options regarding comments.

The default style is `no_blank`.

Violation

```
library ieee;

  use ieee.std_logic_1164.all;

  use ieee.std_logic_unsigned.all;
```

Fix

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;
```

15.28.8 library_008

This rule checks the indent of the **use** keyword.

Violation

```
library ieee;
use ieee.std_logic_1164.all;
     use ieee.std_logic_unsigned.all;
```

Fix

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;
```

15.28.9 library_009

This rule checks alignment of comments above library use statements.

Violation

```
library ieee;
-- Use standard logic library
  use ieee.std_logic_1164.all;
```

Fix

```
library ieee;
  -- Use standard logic library
  use ieee.std_logic_1164.all;
```

15.28.10 library_010

This rule checks the **library** keyword is on it's own line.

Violation

```
context c1 is library ieee; use ieee.std_logic_1164.all; end context c1;
```

Fix

```
context c1 is
  library ieee; use ieee.std_logic_1164.all; end context c1;
```

15.28.11 library_011

This rule checks the **use** keyword is on it's own line.

Violation

```
context c1 is library ieee; use ieee.std_logic_1164.all; end context c1;
```

Fix

```
context c1 is library ieee;
    use ieee.std_logic_1164.all; end context c1;
```

15.29 Loop Statement Rules

15.29.1 loop_statement_300

This rule checks the indentation of the **loop** keyword.

Violation

```
fifo_proc : process () is
begin

    loop

  end loop;

end process;
```

Fix

```
fifo_proc : process () is
begin

  loop

  end loop;

end process;
```

15.30 Package Rules

15.30.1 package_001

This rule checks the indent of the package declaration.

Violation

```
library ieee;

  package FIFO_PKG is
```

Fix

```
library ieee;

package FIFO_PKG is
```

15.30.2 package_002

This rule checks for a single space between **package** and **is** keywords.

Violation

```
package    FIFO_PKG    is
```

Fix

```
package FIFO_PKG is
```

15.30.3 package_003

This rule checks for blank lines or comments above the **package** keyword.

Refer to Configuring Previous Line Rules for options.

The default style is `no_code`.

Violation

```
library ieee;
package FIFO_PKG is
```

Fix

```
library ieee;

package FIFO_PKG is
```

15.30.4 package_004

This rule checks the package keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PACKAGE FIFO_PKG is
```

Fix

```
package FIFO_PKG is
```

15.30.5 package_005

This rule checks the **is** keyword is on the same line as the **package** keyword.

Violation

```
package FIFO_PKG
is
```

Fix

```
package FIFO_PKG is
```

15.30.6 package_006

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END package fifo_pkg;
```

Fix

```
end package fifo_pkg;
```

15.30.7 package_007

This rule checks for the **package** keyword on the end package declaration.

Refer to the section Configuring Optional Items for options.

Violation

```
end FIFO_PKG;
```

Fix

```
end package FIFO_PKG;
```

15.30.8 package_008

This rule checks the package name has proper case on the end package declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end package FIFO_PKG;
```

Fix

```
end package fifo_pkg;
```

15.30.9 package_009

This rule checks for a single space between the **end** and **package** keywords and package name.

Violation

```
end   package   FIFO_PKG;
```

Fix

```
end package FIFO_PKG;
```

15.30.10 package_010

This rule checks the package name has proper case in the package declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
package FIFO_PKG is
```

Fix

```
package fifo_pkg is
```

15.30.11 package_011

This rule checks for a blank line below the **package** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
package FIFO_PKG is
  constant width : integer := 32;
```

Fix

```
package FIFO_PKG is

  constant width : integer := 32;
```

15.30.12 package_012

This rule checks for blank lines or comments above the **end package** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
  constant depth : integer := 512;
end package FIFO_PKG;
```

Fix

```
  constant depth : integer := 512;

end package FIFO_PKG;
```

15.30.13 package_013

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
package fifo_pkg IS
```

Fix

```
package fifo_pkg is
```

15.30.14 package_014

This rule checks the package name exists on the same line as the **end package** keywords.

Refer to the section Configuring Optional Items for options.

Violation

```
end package;
```

Fix

```
end package fifo_pkg;
```

15.30.15 package_015

This rule checks the indent of the end package declaration.

Violation

```
package FIFO_PKG is

   end package fifo_pkg;
```

Fix

```
package fifo_pkg is

end package fifo_pkg;
```

15.30.16 package_016

This rule checks for valid suffixes on package identifiers. The default package suffix is *_pkg*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
package foo is
```

Fix

```
package foo_pkg is
```

15.30.17 package_017

This rule checks for valid prefixes on package identifiers. The default package prefix is *pkg_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
package foo is
```

Fix

```
package pkg_foo is
```

15.30.18 package_018

This rule checks the **package** keyword in the **end package** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end PACKAGE fifo_pkg;
```

Fix

```
end package fifo_pkg;
```

15.30.19 package_019

This rule checks the identifiers for all declarations are aligned in the package declarative region.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable var1 : natural;
signal sig1 : natural;
constant c_period : time;
```

Fix

```
variable var1     : natural;
signal   sig1     : natural;
constant c_period : time;
```

15.30.20 package_400

This rule checks the colons are in the same column for all declarations in the package declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
package my_package is

  signal   wr_en : std_logic;
  signal   rd_en    : std_logic;
  constant c_period : time;

end package my_package;
```

Fix

```
package my_package is

  signal   wr_en    : std_logic;
  signal   rd_en    : std_logic;
  constant c_period : time;

end package my_package;
```

15.30.21 package_401

This rule checks the alignment of inline comments in the package declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
package my_package is

  signal   wr_en    : std_logic;  -- Comment 1
  signal   rd_en    : std_logic;      -- Comment 2
  constant c_period : time; -- Comment 3

end package my_package;
```

Fix

```
package my_package is

  signal   wr_en    : std_logic; -- Comment 1
  signal   rd_en    : std_logic; -- Comment 2
  constant c_period : time;      -- Comment 3

end package my_package;
```

15.31 Package Body Rules

15.31.1 package_body_001

This rule checks the **is** keyword is on the same line as the **package** keyword.

Violation

```
package body FIFO_PKG
is
```

Fix

```
package body FIFO_PKG is
```

15.31.2 package_body_002

This rule checks for the optional **package body** keywords on the end package body declaration.

Refer to the section Configuring Optional Items for options.

Violation

```
end FIFO_PKG;
```

Fix

```
end package body FIFO_PKG;
```

15.31.3 package_body_003

This rule checks the package name exists in the closing of the package body declaration.

Refer to the section Configuring Optional Items for options.

Violation

```
end package body;
```

Fix

```
end package body fifo_pkg;
```

15.31.4 package_body_100

This rule checks for a single space between **package**, **body** and **is** keywords.

Violation

```
package    body   FIFO_PKG   is
```

Fix

```
package body FIFO_PKG is
```

15.31.5 package_body_101

This rule checks for a single space between the **end**, **package** and **body** keywords and package name.

Violation

```
end    package    body     FIFO_PKG;
```

Fix

```
end package   body   FIFO_PKG;
```

15.31.6 package_body_200

This rule checks for blank lines or comments above the **package** keyword.

Refer to Configuring Previous Line Rules for options.

Violation

```
library ieee;
package body FIFO_PKG is
```

Fix

```
library ieee;

package body FIFO_PKG is
```

15.31.7 package_body_201

This rule checks for a blank line below the **package** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
package body FIFO_PKG is
  constant width : integer := 32;
```

Fix

```
package body FIFO_PKG is

  constant width : integer := 32;
```

15.31.8 package_body_202

This rule checks for blank lines or comments above the **end** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
  constant depth : integer := 512;
end package body FIFO_PKG;
```

Fix

```
  constant depth : integer := 512;

end package body FIFO_PKG;
```

15.31.9 package_body_203

This rule checks for a blank line below the **end package** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
end package body FIFO_PKG;
library ieee;
```

Fix

```
end package body FIFO_PKG;
```

15.31.10 package_body_300

This rule checks the indent of the package body keyword.

Violation

```
library ieee;

  package body FIFO_PKG is
```

Fix

```
library ieee;

package body FIFO_PKG is
```

15.31.11 package_body_301

This rule checks the indent of the end package declaration.

Violation

```
package body FIFO_PKG is

   end package body fifo_pkg;
```

Fix

```
package body fifo_pkg is

end package body fifo_pkg;
```

15.31.12 package_body_400

This rule checks the identifiers for all declarations are aligned in the package body declarative region.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable    var1 : natural;
constant  c_period : time;
```

Fix

```
variable var1      : natural;
constant c_period : time;
```

15.31.13 package_body_500

This rule checks the **package** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PACKAGE body FIFO_PKG is
```

Fix

```
package body FIFO_PKG is
```

15.31.14 package_body_501

This rule checks the **body** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
package BODY FIFO_PKG is
```

Fix

```
package body FIFO_PKG is
```

15.31.15 package_body_502

This rule checks the package name has proper case in the package declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
package body FIFO_PKG is
```

Fix

```
package body fifo_pkg is
```

15.31.16 package_body_503

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
package fifo_pkg IS
```

Fix

```
package fifo_pkg is
```

15.31.17 package_body_504

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END package fifo_pkg;
```

Fix

```
end package fifo_pkg;
```

15.31.18 package_body_505

This rule checks the **package** keyword in the **end package body** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end PACKAGE body fifo_pkg;
```

Fix

```
end package body fifo_pkg;
```

15.31.19 package_body_506

This rule checks the **body** keyword in the **end package body** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end package BODY fifo_pkg;
```

Fix

```
end package body fifo_pkg;
```

15.31.20 package_body_507

This rule checks the package name has proper case on the end package declaration.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end package body FIFO_PKG;
```

Fix

```
end package fifo_pkg;
```

15.31.21 package_body_600

This rule checks for valid suffixes on package body identifiers. The default package suffix is *_pkg*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
package body foo is
```

Fix

```
package body foo_pkg is
```

15.31.22 package_body_601

This rule checks for valid prefixes on package body identifiers. The default package prefix is *pkg_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
package body foo is
```

Fix

```
package body pkg_foo is
```

15.32 Port Rules

15.32.1 port_001

This rule checks for a blank line above the **port** keyword.

Violation

```
entity FIFO is

  port (
```

Fix

```
entity FIFO is
  port (
```

15.32.2 port_002

This rule checks the indent of the **port** keyword.

Violation

```
entity FIFO is
port (
```

Fix

```
entity FIFO is
  port (
```

15.32.3 port_003

This rule checks for a single space after the **port** keyword and (.

Violation

```
port    (

port(
```

Fix

```
port (

port (
```

15.32.4 port_004

This rule checks the indent of port declarations.

Violation

```
port (
WR_EN     : in     std_logic;
     RD_EN     : in     std_logic;
  OVERFLOW : out    std_logic
);
```

Fix

```
port (
  WR_EN     : in     std_logic;
  RD_EN     : in     std_logic;
  OVERFLOW : out    std_logic
);
```

15.32.5 port_005

This rule checks for a single space after the colon.

Violation

```
port (
  WR_EN     : in     std_logic;
  RD_EN     :   in     std_logic;
  OVERFLOW :out    std_logic;
  DATA      :inout std_logic
);
```

Fix

```
port (
  WR_EN     : in     std_logic;
  RD_EN     : in     std_logic;
  OVERFLOW : out    std_logic;
  DATA      : inout std_logic
);
```

15.32.6 port_006

This rule has been depricated and it's function was include in rule **port_005**.

15.32.7 port_007

This rule checks for four spaces after the **in** keyword.

Violation

```
port (
  WR_EN    : in std_logic;
  RD_EN    : in          std_logic;
  OVERFLOW : out    std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic
);
```

15.32.8 port_008

This rule checks for three spaces after the **out** keyword.

Violation

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic
);
```

15.32.9 port_009

This rule checks for a single space after the **inout** keyword.

Violation

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  DATA     : inout    std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  DATA     : inout std_logic
);
```

15.32.10 port_010

This rule checks port names are uppercase.

Violation

```
port (
  wr_en     : in    std_logic;
  rd_en     : in    std_logic;
  OVERFLOW  : out   std_logic;
  underflow : out   std_logic
);
```

Fix

```
port (
  WR_EN     : in    std_logic;
  RD_EN     : in    std_logic;
  OVERFLOW  : out   std_logic;
  UNDERFLOW : out   std_logic
);
```

15.32.11 port_011

This rule checks for valid prefixes on port identifiers. The default port prefixes are: *i_*, *o_*, *io_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
port (
  wr_en    : in    std_logic;
  rd_en    : in    std_logic;
  overflow : out   std_logic;
  data     : inout std_logic
);
```

Fix

```
port (
  i_wr_en    : in    std_logic;
  i_rd_en    : in    std_logic;
  o_overflow : out   std_logic;
  io_data    : inout std_logic
);
```

15.32.12 port_012

This rule checks for default assignments on port declarations.

This rule is defaulted to not fixable and can be overridden with a configuration to remove the default assignments.

Violation

```
port (
  I_WR_EN    : in    std_logic := '0';
  I_RD_EN    : in    std_logic := '0';
  O_OVERFLOW : out   std_logic;
  IO_DATA    : inout std_logic := (others => 'Z')
);
```

Fix

```
port (
  I_WR_EN    : in    std_logic;
  I_RD_EN    : in    std_logic;
  O_OVERFLOW : out   std_logic;
  IO_DATA    : inout std_logic
);
```

15.32.13 port_013

This rule checks for multiple ports declared on a single line.

Violation

```
port (
  WR_EN    : in    std_logic;RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;DATA     : inout std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.14 port_014

This rule checks the closing parenthesis of the port map is on a line by itself.

Violation

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.15 port_015

This rule checks the indent of the closing parenthesis for port maps.

Violation

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
  );
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.16 port_016

This rule checks for a port definition on the same line as the **port** keyword.

Violation

```
port (WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.17 port_017

This rule checks the **port** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PORT (
```

Fix

```
port (
```

15.32.18 port_018

This rule checks the port type has proper case if it is a VHDL keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
port (
  WR_EN    : in    STD_LOGIC;
  RD_EN    : in    std_logic;
  OVERFLOW : out   t_OVERFLOW;
  DATA     : inout STD_LOGIC_VECTOR(31 downto 0)
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   t_OVERFLOW;
  DATA     : inout std_logic_vector(31 downto 0)
);
```

15.32.19 port_019

This rule checks the port direction has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
port (
  WR_EN    : IN    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : OUT   std_logic;
  DATA     : INOUT std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.20 port_020

This rule checks for at least one space before the colon.

Violation

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW: out   std_logic;
  DATA     : inout std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.21 port_021

This rule checks the **port** keyword is on the same line as the (.

Violation

```
port
(
```

Fix

```
port (
```

15.32.22 port_022

This rule checks for blank lines after the **port** keyword.

Violation

```
port (


  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW: out   std_logic;
  DATA     : inout std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.23 port_023

This rule checks for missing modes in port declarations.

Note: This must be fixed by the user. VSG makes no assumption on the direction of the port.

Violation

```
port (
  WR_EN    : std_logic;
  RD_EN    : std_logic;
  OVERFLOW : std_logic;
  DATA     : inout std_logic
);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.24 port_024

This rule checks for blank lines before the close parenthesis in port declarations.

Violation

```
port (
  WR_EN    : std_logic;
  RD_EN    : std_logic;
  OVERFLOW : std_logic;
  DATA     : inout std_logic



);
```

Fix

```
port (
  WR_EN    : in    std_logic;
  RD_EN    : in    std_logic;
  OVERFLOW : out   std_logic;
  DATA     : inout std_logic
);
```

15.32.25 port_025

This rule checks for valid suffixes on port identifiers. The default port suffixes are *_i*, *_o*, *_io*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
port (
  wr_en    : in    std_logic;
  rd_en    : in    std_logic;
  overflow : out   std_logic;
  data     : inout std_logic
);
```

Fix

```
port (
  wr_en_i    : in    std_logic;
  rd_en_i    : in    std_logic;
  overflow_o : out   std_logic;
  data_io    : inout std_logic
);
```

15.32.26 port_026

This rule checks for multiple identifiers on port declarations.

Any comments are not replicated.

Violation

```
port (
  wr_en, rd_en : in     std_logic;   -- Comment
  data      : inout std_logic;
  overflow, empty : out    std_logic -- Other comment
);
```

Fix

```
port (
  wr_en    : in     std_logic;
  rd_en    : in     std_logic;   -- Comment
  data    : inout std_logic
  overflow : out    std_logic;
  empty : out    std_logic -- Other comment
);
```

15.33 Port Map Rules

15.33.1 port_map_001

This rule checks the **port map** keywords have proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PORT MAP (
```

Fix

```
port map (
```

15.33.2 port_map_002

This rule checks the port name is uppercase. Indexes on ports will not be uppercased.

Violation

```
port map (
  wr_en              => wr_en,
  rd_en              => rd_en,
  OVERFLOW           => overflow,
  underflow(c_index) => underflow
);
```

Fix

```
port map (
  WR_EN                => wr_en,
  RD_EN                => rd_en,
  OVERFLOW             => overflow,
  UNDERFLOW(c_index) => underflow
);
```

15.33.3 port_map_003

This rule checks the (is on the same line as the **port map** keywords.

Violation

```
port map
(
  WR_EN    => WR_EN,
  RD_EN    => RD_EN,
  OVERFLOW => OVERFLOW
);
```

Fix

Use explicit port mapping.

```
port map (
  WR_EN    => WR_EN,
  RD_EN    => RD_EN,
  OVERFLOW => OVERFLOW
);
```

15.33.4 port_map_004

This rule checks the closing) for the port map is on it's own line.

Violation

```
port map (
  WR_EN => wr_en);
```

Fix

```
port map (
  WR_EN => wr_en
);
```

15.33.5 port_map_005

This rule checks for a port assignment on the same line as the **port map** keyword.

Violation

```
port map (WR_EN    => wr_en,
  RD_EN    => rd_en,
  OVERFLOW => overflow
);
```

Fix

```
port map (
  WR_EN    => wr_en,
  RD_EN    => rd_en,
  OVERFLOW => overflow
);
```

15.33.6 port_map_007

This rule checks for a single space after the **=>** operator in port maps.

Violation

```
U_FIFO : FIFO
  port map (
    WR_EN    =>   wr_en,
    RD_EN    =>rd_en,
    OVERFLOW =>     overflow
  );
```

Fix

```
U_FIFO : FIFO
  port map (
    WR_EN    => wr_en,
    RD_EN    => rd_en,
    OVERFLOW => overflow
  );
```

15.33.7 port_map_008

This rule checks for positional ports. Positional ports are subject to problems when the position of the underlying component changes.

Violation

```
port map (
  WR_EN, RD_EN, OVERFLOW
);
```

Fix

Use explicit port mapping.

```
port map (
  WR_EN    => WR_EN,
  RD_EN    => RD_EN,
  OVERFLOW => OVERFLOW
);
```

15.33.8 port_map_009

This rule checks multiple port assignments on the same line.

Violation

```
port map (
  WR_EN => w_wr_en, RD_EN => w_rd_en,
  OVERFLOW => w_overflow
);
```

Fix

```
port map (
  WR_EN => w_wr_en,
  RD_EN => w_rd_en,
  OVERFLOW => w_overflow
);
```

15.34 Procedure Rules

There are three forms a procedure: with parameters, without parameters, and a package declaration:

with parameters

```
procedure average_samples (
   constant a : in integer;
   signal b : in std_logic;
   variable c : in std_logic_vector(3 downto 0);
   signal d : out std_logic) is
begin
end procedure average_samples;
```

without parameters

```
procedure average_samples is
begin
end procedure average_samples;
```

package declaration

```
procedure average_samples;

procedure average_samples (
   constant a : in integer;
   signal b : in std_logic;
```

(continues on next page)

(continued from previous page)

```
   variable c : in std_logic_vector(3 downto 0);
   signal d : out std_logic);
```

15.34.1 procedure_001

This rule checks the indent of the **procedure** keyword.

Violation

```
  procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
end procedure average_samples;
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
end procedure average_samples;
```

15.34.2 procedure_002

This rule checks the indent of the **begin** keyword.

Violation

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
  begin
end procedure average_samples;
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
end procedure average_samples;
```

15.34.3 procedure_003

This rule checks the indent of the **end** keyword.

Violation

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
  end procedure average_samples;
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
end procedure average_samples;
```

15.34.4 procedure_004

This rule checks the indent of parameters.

Violation

```
procedure average_samples (
constant a : in integer;
    signal b : in std_logic;
   variable c : in std_logic_vector(3 downto 0);
 signal d : out std_logic ) is
begin
end procedure average_samples;
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
begin
end procedure average_samples;
```

15.34.5 procedure_005

This rule checks the indent of line between the **is** and **begin** keywords

Violation

```
procedure average_samples (
  constant a : in integer;
  signal d : out std_logic ) is
variable var_1 : integer;
    variable var_1 : integer;
begin
end procedure average_samples;
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal b : in std_logic;
  variable c : in std_logic_vector(3 downto 0);
  signal d : out std_logic ) is
  variable var_1 : integer;
  variable var_1 : integer;
begin
end procedure average_samples;
```

15.34.6 procedure_006

This rule checks the indent of the closing parenthesis if it is on it's own line.

Violation

```
procedure average_samples (
  constant a : in integer;
  signal d : out std_logic
  ) is
```

Fix

```
procedure average_samples (
  constant a : in integer;
  signal d : out std_logic
) is
```

15.34.7 procedure_007

This rule checks for consistent capitalization of procedure names.

Violation

```
architecture rtl of entity1 is

  procedure average_samples (
    constant a : in integer;
    signal d : out std_logic
  ) is
```

(continues on next page)

(continued from previous page)

```
begin

  proc1 : process () is
  begin

    Average_samples();

  end process proc1;

end architecture rtl;
```

Fix

```
architecture rtl of entity1 is

  procedure average_samples (
    constant a : in integer;
    signal d : out std_logic
  ) is

begin

  proc1 : process () is
  begin

    average_samples();

  end process proc1;

end architecture RTL;
```

15.34.8 procedure_008

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END;

End procedure proc;
```

Fix

```
end;

end procedure proc;
```

15.34.9 procedure_009

This rule checks the **procedure** keyword in the **end procedure** has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end PROCEDURE;

end Procedure proc;
```

Fix

```
end procedure;

end procedure proc;
```

15.34.10 procedure_010

This rule checks the identifiers for all declarations are aligned in the procedure declarative part.

Refer to the section Configuring Identifier Alignment Rules for information on changing the configurations.

Violation

```
variable var1 : natural;
signal sig1 : natural;
constant c_period : time;
```

Fix

```
variable var1     : natural;
signal   sig1     : natural;
constant c_period : time;
```

15.35 Procedure Call Rules

These rules handle **procedure_call_statement** and **concurrent_procedure_call_statement** elements.

15.35.1 procedure_call_001

This rule checks for labels on procedure call statements. Labels on procedure calls are optional and do not provide additional information.

Violation

```
WR_EN_OUTPUT : WR_EN(parameter);
```

Fix

```
WR_EN(parameter);
```

15.35.2 procedure_call_002

This rule checks for labels on concurrent procedure call statements. Labels on procedure calls are optional and do not provide additional information.

Violation

```
WR_EN_OUTPUT : WR_EN(parameter);
```

Fix

```
WR_EN(parameter);
```

15.35.3 procedure_call_100

This rule checks for a single space between the following block elements: label, label colon, **postponed** keyword and the *procedure* name.

Violation

```
procedure_label    :     postponed    WR_EN(parameter);
```

Fix

```
procedure_label : postponed WR_EN(parameter);
```

15.35.4 procedure_call_300

This rule checks the indent of the procedure_call label.

Violation

```
a <= b;

  procedure_label : WR_EN(parameter);
```

Fix

```
a <= b;

procedure_label : WR_EN(parameter);
```

15.35.5 procedure_call_301

This rule checks the indent of the **postponed** keyword if it exists..

Violation

```
a <= b;

  postponed WR_EN(parameter);
```

Fix

```
a <= b;

postponed WR_EN(parameter);
```

15.35.6 procedure_call_302

This rule checks the indent of the *procedure* name.

Violation

```
a <= b;

  WR_EN(parameter);
```

Fix

```
a <= b;

WR_EN(parameter);
```

15.35.7 procedure_call_500

This rule checks the label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PROCEDURE_CALL_LABEL : WR_EN(paremeter);
```

Fix

```
procedure_call_label : WR_EN(paremeter);
```

15.35.8 procedure_call_501

This rule checks the **postponed** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
POSTPONED WR_EN(parameter)
```

Fix

```
postponed WR_EN(parameter)
```

15.36 Process Rules

15.36.1 process_001

This rule checks the indent of the process declaration.

Violation

```
architecture rtl of fifo is

begin

proc_a : process (rd_en, wr_en, data_in, data_out,
```

Fix

```
architecture rtl of fifo is

begin

  proc_a : process (rd_en, wr_en, data_in, data_out,
```

15.36.2 process_002

This rule checks for a single space after the **process** keyword.

Violation

```
proc_a : process(rd_en, wr_en, data_in, data_out,

proc_a : process     (rd_en, wr_en, data_in, data_out,
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,

proc_a : process (rd_en, wr_en, data_in, data_out,
```

15.36.3 process_003

This rule checks the indent of the **begin** keyword.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
  begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.4 process_004

This rule checks the **begin** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
BEGIN
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.5 process_005

This rule checks the **process** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
proc_a : PROCESS (rd_en, wr_en, data_in, data_out,
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
```

15.36.6 process_006

This rule checks the indent of the **end process** keywords.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin

  end process proc_a;
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin

end process proc_a;
```

15.36.7 process_007

This rule checks for a single space after the **end** keyword.

Violation

```
end   process proc_a;
```

Fix

```
end process proc_a;
```

15.36.8 process_008

This rule checks the **end** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
END process proc_a;
```

Fix

```
end process proc_a;
```

15.36.9 process_009

This rule checks the **process** keyword has proper case in the **end process** line.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end PROCESS proc_a;
```

Fix

```
end process proc_a;
```

15.36.10 process_010

This rule checks the **begin** keyword is on it's own line.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.11 process_011

This rule checks for a blank line below the **end process** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
end process proc_a;
wr_en <= wr_en;
```

Fix

```
end process proc_a;

wr_en <= wr_en;
```

15.36.12 process_012

This rule checks for the existence of the **is** keyword.

Refer to the section Configuring Optional Items for options.

Violation

```
proc_a : process
begin
end process;

proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 )
begin
end process;
```

Fix

```
proc_a : process is
begin
end process;


proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
end process;
```

15.36.13 process_013

This rule checks the **is** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) IS
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.14 process_014

This rule checks for a single space before the **is** keyword.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
```

(continues on next page)

(continued from previous page)

```
                  )      is
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.15 process_015

This rule checks for blank lines or comments above the **process** declaration.

Refer to the section Configuring Blank Lines for options regarding comments.

The default style is `no_code`.

Violation

```
-- This process performs FIFO operations.
proc_a : process (rd_en, wr_en, data_in, data_out,

wr_en <= wr_en;
proc_a : process (rd_en, wr_en, data_in, data_out,
```

Fix

```
-- This process performs FIFO operations.
proc_a : process (rd_en, wr_en, data_in, data_out,

wr_en <= wr_en;

proc_a : process (rd_en, wr_en, data_in, data_out,
```

15.36.16 process_016

This rule checks the process has a label.

Violation

```
process (rd_en, wr_en, data_in, data_out,
         rd_full, wr_full
        ) is
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.17 process_017

This rule checks the process label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
PROC_A : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.18 process_018

This rule checks the **end process** line has a label. The closing label will be added if the opening process label exists.

Refer to the section Configuring Optional Items for options.

Violation

```
end process;
```

Fix

```
end process proc_a;
```

15.36.19 process_019

This rule checks the **end process** label has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
end process PROC_A;
```

Fix

```
end process proc_a;
```

15.36.20 process_020

This rule checks the indentation of multiline sensitivity lists.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                     rd_full, wr_full,
            overflow, underflow
                 ) is begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full,
                  overflow, underflow
                 ) is
begin
```

15.36.21 process_021

This rule checks for blank lines above the **begin** keyword if there are no process declarative items.

Refer to Configuring Blank Lines for options.

Violation

```
proc_a : process

begin


proc_a : process (rd_en, wr_en)

begin


proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is



begin
```

Fix

```
proc_a : process
begin


proc_a : process (rd_en, wr_en)
begin

```

(continues on next page)

(continued from previous page)

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.22 process_022

This rule checks for a blank line below the **begin** keyword.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
  rd_en <= '0';
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin

  rd_en <= '0';
```

15.36.23 process_023

This rule checks for a blank line above the **end process** keyword.

Refer to Configuring Blank Lines for options.

Violation

```
  wr_en <= '1';
end process proc_a;
```

Fix

```
  wr_en <= '1';

end process proc_a;
```

15.36.24 process_024

This rule checks for a single space after the process label.

Violation

```
proc_a: process (rd_en, wr_en, data_in, data_out,
                 rd_full, wr_full
                ) is
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.25 process_025

This rule checks for a single space after the colon and before the **process** keyword.

Violation

```
proc_a :process (rd_en, wr_en, data_in, data_out,
                 rd_full, wr_full
                ) is begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
begin
```

15.36.26 process_026

This rule checks for blank lines above the first declarative line, if it exists.

Refer to Configuring Blank Lines for options.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is
  -- Keep track of the number of words in the FIFO
  variable word_count : integer;
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is

```

(continues on next page)

(continued from previous page)

```
  -- Keep track of the number of words in the FIFO
  variable word_count : integer;
begin
```

15.36.27 process_027

This rule checks for blank lines above the **begin** keyword if a declarative item exists.

Refer to Configuring Blank Lines for options.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is

  -- Keep track of the number of words in the FIFO
  variable word_count : integer;
begin
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 ) is

  -- Keep track of the number of words in the FIFO
  variable word_count : integer;

begin
```

15.36.28 process_028

This rule checks the alignment of the closing parenthesis of a sensitivity list. Parenthesis on multiple lines should be in the same column.

Violation

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                    )
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 )
```

15.36.29 process_029

This rule checks for the format of clock definitions in clock processes. The rule can be set to enforce **event** definition:

```
if (clk'event and clk = '1') then
```

..or **edge** definition:

```
if (rising_edge(clk)) then
```

event configuration

Note: This is the default configuration.

Violation

```
if (rising_edge(clk)) then

if (falling_edge(clk)) then
```

Fix

```
if (clk'event and clk = '1') then

if (clk'event and clk = '0') then
```

edge configuration

Note: Configuration this by setting the *'clock'* attribute to *'edge'*

```
{
  "rule":{
    "process_029":{
       "clock":"edge"
    }
  }
}
```

Violation

```
if (clk'event and clk = '1') then

if (clk'event and clk = '0') then
```

Fix

```
if (rising_edge(clk)) then

if (falling_edge(clk)) then
```

15.36.30 process_030

This rule checks for a single signal per line in a sensitivity list that is not the last one. The sensitivity list is required by the compiler, but provides no useful information to the reader. Therefore, the vertical spacing of the sensitivity list should be minimized. This will help with code readability.

Note: This rule is left to the user to fix.

Violation

```
proc_a : process (rd_en,
                  wr_en,
                  data_in,
                  data_out,
                  rd_full,
                  wr_full
                 )
```

Fix

```
proc_a : process (rd_en, wr_en, data_in, data_out,
                  rd_full, wr_full
                 )
```

15.36.31 process_031

This rule checks for alignment of identifiers in the process declarative region.

Violation

```
proc_1 : process(all) is

 variable     var1 : boolean;
 constant   cons1 : integer;
 file            file1   : load_file_file open read_mode is load_file_name;

begin

end process proc_1;
```

Fix

```
proc_1 : process(all) is

 variable var1 : boolean;
 constant cons1 : integer;
 file     file1  : load_file_file open read_mode is load_file_name;

begin

end process proc_1;
```

15.36.32 process_032

This rule checks the process label is on the same line as the process keyword.

Violation

```
proc_1 :

process(all) is
```

Fix

```
proc_1 : process(all) is
```

15.36.33 process_033

This rule checks the colons are in the same column for all declarations in the process declarative part. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
variable var1 : natural;
variable var2  : natural;
constant c_period : time;
file my_test_input : my_file_type;
```

Fix

```
variable var1        : natural;
variable var2        : natural;
constant c_period    : time;
file my_test_input : my_file_type;
```

15.36.34 process_034

This rule aligns inline comments between the end of the process sensitivity list and the process **begin** keyword. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
proc_1 : process () is

   variable counter : integer range 0 to 31;      -- Counts the number of frames␣
↪received
   variable width   : natural range 0 to 255; -- Keeps track of the data word size

   variable size    : natural range 0 to 7; -- Keeps track of the frame size

begin
```

Fix

```
proc_1 : process () is

   variable counter : integer range 0 to 31;  -- Counts the number of frames received
   variable width   : natural range 0 to 255; -- Keeps track of the data word size

   variable size    : natural range 0 to 7;   -- Keeps track of the frame size

begin
```

15.36.35 process_035

This rule checks the alignment of inline comments between the process begin and end process lines. Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
proc_1: process () is
begin

  a <= '1';    -- Assert
  b <= '0';        -- Deassert
  c <= '1'; -- Enable

end process proc_1;
```

Fix

```
proc_1: process () is
begin

  a <= '1'; -- Assert
  b <= '0'; -- Deassert
  c <= '1'; -- Enable

end process proc_1;
```

15.36.36 process_036

This rule checks for valid prefixes on process labels. The default prefix is *proc_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
main: process () is
```

Fix

```
proc_main: process () is
```

15.36.37 process_600

This rule checks for valid suffixes on process labels. The default suffix is *_proc*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
main: process () is
```

Fix

```
main_proc: process () is
```

15.37 Report Statement Rules

15.37.1 report_statement_001

This rule removes labels on report_statement_statements.

Violation

```
REPORT_LABEL : report "FIFO width is limited to 16 bits.";
```

Fix

```
REPORT_LABEL : report "FIFO width is limited to 16 bits.";
```

15.37.2 report_statement_002

This rule checks the **severity** keyword is on it's own line.

Violation

```
report "FIFO width is limited to 16 bits." severity FAILURE;
```

Fix

```
report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.37.3 report_statement_100

This rule checks for a single space after the **report** keyword.

Violation

```
report      "FIFO width is limited to 16 bits.";
```

Fix

```
report "FIFO width is limited to 16 bits.";
```

15.37.4 report_statement_101

This rule checks for a single space after the **severity** keyword.

Violation

```
report FIFO width is limited to 16 bits."
  severity    FAILURE;
```

Fix

```
report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.37.5 report_statement_300

This rule checks indent of multiline report statements.

Violation

```
report FIFO width is limited to 16 bits."
        severity FAILURE;
```

Fix

```
report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.37.6 report_statement_400

This rule checks the alignment of the report expressions.

Note: There is a configuration option **alignment** which changes the indent location of multiple lines.

alignment set to 'report' (Default)

Violation

```
report "FIFO width is limited" &
" to 16 bits."
  severity FAILURE;
```

Fix

```
report "FIFO width is limited" &
       " to 16 bits."
  severity FAILURE;
```

alignment set to 'left'

Violation

```
report "FIFO width is limited" &
" to 16 bits."
  severity FAILURE;
```

Fix

```
report "FIFO width is limited" &
    " to 16 bits."
  severity FAILURE;
```

15.37.7 report_statement_500

This rule checks the **report** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
REPORT "FIFO width is limited to 16 bits."
  severity FAILURE;
```

```
report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.37.8 report_statement_501

This rule checks the **severity** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
report "FIFO width is limited to 16 bits."
  SEVERITY FAILURE;
```

```
report "FIFO width is limited to 16 bits."
  severity FAILURE;
```

15.38 Range Rules

These rules cover the range definitions in signals, constants, ports and other cases where ranges are defined.

15.38.1 range_001

This rule checks the case of the **downto** keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
signal sig1 : std_logic_vector(3 DOWNTO 0);
signal sig2 : std_logic_vector(16 downTO 1);
```

Fix

```
signal sig1 : std_logic_vector(3 downto 0);
signal sig2 : std_logic_vector(16 downTO 1);
```

15.38.2 range_002

This rule checks the case of the **to** keyword.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
signal sig1 : std_logic_vector(3 TO 0);
signal sig2 : std_logic_vector(16 tO 1);
```

Fix

```
signal sig1 : std_logic_vector(3 to 0);
signal sig2 : std_logic_vector(16 to 1);
```

15.39 Sequential Rules

15.39.1 sequential_001

This rule checks the indent of sequential statements.

Violation

```
begin

    wr_en <= '1';
rd_en <= '0';
```

Fix

```
begin

  wr_en <= '1';
  rd_en <= '0';
```

15.39.2 sequential_002

This rule checks for a single space after the **<=** operator.

Violation

```
wr_en <=      '1';
rd_en <='0';
```

Fix

```
wr_en <= '1';
rd_en <= '0';
```

15.39.3 sequential_003

This rule checks for at least a single space before the **<=** operator.

Violation

```
wr_en<= '1';
rd_en    <= '0';
```

Fix

```
wr_en <= '1';
rd_en    <= '0';
```

15.39.4 sequential_004

This rule checks the alignment of multiline sequential statements.

Violation

```
overflow <= wr_en and
  rd_en;
```

Fix

```
overflow <= wr_en and
            rd_en;
```

15.39.5 sequential_005

This rule checks the alignment of the **<=** operators over consecutive sequential lines.

Following extra configurations are supported:

- `if_control_statements_end_group`,
- `case_control_statements_end_group`.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
wr_en <= '1';
rd_en   <= '0';
```

Fix

```
wr_en <= '1';
rd_en <= '0';
```

15.39.6 sequential_006

This rule checks for comments within multiline sequential statements.

Violation

```
overflow <= wr_en and
 --         rd_address(0)
            rd_en;
```

Fix

```
overflow <= wr_en and
            rd_en;
```

15.39.7 sequential_007

This rule checks for code after a sequential assignment.

Violation

```
a <= '0'; b <= '1'; c <= '0'; -- comment
```

Fix

```
a <= '0';
b <= '1';
c <= '0'; -- comment
```

15.40 Signal Rules

15.40.1 signal_001

This rule checks the indent of signal declarations.

Violation

```
architecture rtl of fifo is

signal wr_en : std_logic;
     signal rd_en : std_logic;

begin
```

Fix

```
architecture rtl of fifo is

  signal wr_en : std_logic;
  signal rd_en : std_logic;

begin
```

15.40.2 signal_002

This rule checks the **signal** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
SIGNAL wr_en : std_logic;
```

Fix

```
signal wr_en : std_logic;
```

15.40.3 signal_003

This rule was depricated and replaced with rules:

- function_015
- package_019

- procedure_010
- architecture_029

15.40.4 signal_004

This rule checks the signal name has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
signal WR_EN : std_logic;
```

Fix

```
signal wr_en : std_logic;
```

15.40.5 signal_005

This rule checks for a single space after the colon.

Violation

```
signal wr_en :    std_logic;
signal rd_en :std_logic;
```

Fix

```
signal wr_en : std_logic;
signal rd_en : std_logic;
```

15.40.6 signal_006

This rule checks for at least a single space before the colon.

Violation

```
signal wr_en: std_logic;
signal rd_en   : std_logic;
```

Fix

```
signal wr_en : std_logic;
signal rd_en   : std_logic;
```

15.40.7 signal_007

This rule checks for default assignments in signal declarations.

Note: This rule is requires the user to remove the default assignments.

Violation

```
signal wr_en : std_logic := '0';
```

Fix

```
signal wr_en : std_logic;
```

15.40.8 signal_008

This rule checks for valid prefixes on signal identifiers. Default signal prefix is *s_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
signal wr_en : std_logic;
signal rd_en : std_logic;
```

Fix

```
signal s_wr_en : std_logic;
signal s_rd_en : std_logic;
```

15.40.9 signal_010

This rule checks the signal type has proper case if it is a VHDL keyword.

Note: This rule is disabled by default.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
signal wr_en    : STD_LOGIC;
signal rd_en    : Std_logic;
signal cs_f     : t_User_Defined_Type;
```

Fix

```
signal wr_en    : std_logic;
signal rd_en    : std_logic;
signal cs_f     : t_User_Defined_Type;
```

15.40.10 signal_011

This rule checks the signal type has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
signal wr_en    : STD_LOGIC;
signal rd_en    : Std_logic;
signal cs_f     : t_User_Defined_Type;
```

Fix

```
signal wr_en    : std_logic;
signal rd_en    : std_logic;
signal cs_f     : t_user_defined_type;
```

15.40.11 signal_012

This rule checks multiple signal declarations on a single line are column aligned.

Note: This rule will only cover two signals on a single line.

Violation

```
signal wr_en, wr_en_f                : std_logic;
signal rd_en_f, rd_en                : std_logic;
signal chip_select, chip_select_f : t_user_defined_type;
```

Fix

```
signal wr_en,       wr_en_f       : std_logic;
signal rd_en_f,     rd_en         : std_logic;
signal chip_select, chip_select_f : t_user_defined_type;
```

15.40.12 signal_014

This rule checks for consistent capitalization of signal names.

Violation

```
architecture rtl of entity1 is

  signal sig1 : std_logic;
  signal sig2 : std_logic;

begin

  proc_name : process (siG2) is
  begin

    siG1 <= '0';

    if (SIG2 = '0') then
      sIg1 <= '1';
    elisif (SiG2 = '1') then
      SIg1 <= '0';
    end if;

  end process proc_name;

end architecture rtl;
```

Fix

```
architecture rtl of entity1 is

  signal sig1 : std_logic;
  signal sig2 : std_logic;

  proc_name : process (sig2) is
  begin

    sig1 <= '0';

    if (sig2 = '0') then
      sig1 <= '1';
    elisif (sig2 = '1') then
      sig1 <= '0';
    end if;

  end process proc_name;

end architecture rtl;
```

15.40.13 signal_015

This rule checks for multiple signal names defined in a single signal declaration. By default, this rule will only flag more than two signal declarations.

Refer to the section Configuring Number of Signals in Signal Declaration for information on changing the default.

Violation

```
signal sig1, sig2
  sig3, sig4,
```

(continues on next page)

(continued from previous page)

```
  sig5
  : std_logic;
```

Fix

```
signal sig1 : std_logic;
signal sig2 : std_logic;
signal sig3 : std_logic;
signal sig4 : std_logic;
signal sig5 : std_logic;
```

15.40.14 signal_016

This rule checks the signal declaration is on a single line.

Violation

```
signal sig1
  : std_logic;

signal sig2 :
  std_logic;
```

Fix

```
signal sig1 : std_logic;

signal sig2 : std_logic;
```

15.40.15 signal_600

This rule checks for valid suffixes on signal identifiers. Default signal suffix is *_s*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
signal wr_en : std_logic;
signal rd_en : std_logic;
```

Fix

```
signal wr_en_s : std_logic;
signal rd_en_s : std_logic;
```

15.41 Source File Rules

15.41.1 source_file_001

This rule checks for the existance of the source file passed to VSG.

Violation

Source file passed to VSG does not exist. This violation will be reported at the command line in the normal output. It will also be reported in the junit file if the –junit option is used.

Fix

Pass correct file name to VSG.

15.42 Subtype Rules

15.42.1 subtype_001

This rule checks for indentation of the **subtype** keyword. Proper indentation enhances comprehension.

The indent amount can be controlled by the **indentSize** attribute on the rule. **indentSize** defaults to 2.

Violation

```
architecture rtl of fifo is

     subtype read_size is range 0 to 9;
subtype write_size is range 0 to 9;

begin
```

Fix

```
architecture rtl of fifo is

  subtype read_size is range 0 to 9;
  subtype write_size is range 0 to 9;

begin
```

15.42.2 subtype_002

This rule checks for consistent capitalization of subtype names.

Violation

```
subtype read_size is range 0 to 9;
subtype write_size is range 0 to 9;

signal read  : READ_SIZE;
signal write : write_size;

constant read_sz  : read_size := 8;
constant write_sz : WRITE_size := 1;
```

Fix

```
subtype read_size is range 0 to 9;
subtype write_size is range 0 to 9;

signal read  : read_size;
signal write : write_size;

constant read_sz  : read_size := 8;
constant write_sz : write_size := 1;
```

15.42.3 subtype_003

This rule was depricated and replaced with rules:

- function_015
- package_019
- procedure_010
- architecture_029

15.42.4 subtype_004

This rule checks for valid prefixes in subtype identifiers. The default new subtype prefix is *st_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
subtype my_subtype is range 0 to 9;
```

Fix

```
subtype st_my_subtype is range 0 to 9;
```

15.42.5 subtype_600

This rule checks for valid suffixes in subtype identifiers. The default new subtype suffix is *_st*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
subtype my_subtype is range 0 to 9;
```

Fix

```
subtype my_subtype_st is range 0 to 9;
```

15.43 Type Rules

15.43.1 type_001

This rule checks the indent of the **type** declaration.

Violation

```
architecture rtl of fifo is

    type state_machine is (idle, write, read, done);

begin
```

Fix

```
architecture rtl of fifo is

  type state_machine is (idle, write, read, done);

begin
```

15.43.2 type_002

This rule checks the **type** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
TYPE state_machine is (idle, write, read, done);
```

Fix

```
type state_machine is (idle, write, read, done);
```

15.43.3 type_003

This rule was depricated and replaced with rules:

- function_015
- package_019

- procedure_010
- architecture_029

15.43.4 type_004

This rule checks the type identifier has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
type STATE_MACHINE is (idle, write, read, done);
```

Fix

```
type state_machine is (idle, write, read, done);
```

15.43.5 type_005

This rule checks the indent of multiline enumerated types.

Violation

```
type state_machine is (
idle,
  write,
read,
   done);
```

Fix

```
type state_machine is (
  idle,
  write,
  read,
  done);
```

15.43.6 type_006

This rule checks for a single space before the **is** keyword.

Violation

```
type state_machine    is (idle, write, read, done);
```

Fix

```
type state_machine is (idle, write, read, done);
```

15.43.7 type_007

This rule checks for a single space after the **is** keyword.

Violation

```
type state_machine is      (idle, write, read, done);
```

Fix

```
type state_machine is (idle, write, read, done);
```

15.43.8 type_008

This rule checks the closing parenthesis of multiline enumerated types is on it's own line.

Violation

```
type state_machine is (
  idle,
  write,
  read,
  done);
```

Fix

```
type state_machine is (
  idle,
  write,
  read,
  done
);
```

15.43.9 type_009

This rule checks for an enumerate type after the open parenthesis on multiline enumerated types.

Violation

```
type state_machine is (idle,
  write,
  read,
  done
);
```

Fix

```
type state_machine is (
  idle,
  write,
```

(continues on next page)

(continued from previous page)

```
  read,
  done
);
```

15.43.10 type_010

This rule checks for blank lines or comments above the **type** declaration.

Refer to Configuring Previous Line Rules for options.

Violation

```
signal wr_en : std_logic;
type state_machine is (idle, write, read, done);
```

Fix

```
signal wr_en : std_logic;

type state_machine is (idle, write, read, done);
```

15.43.11 type_011

This rule checks for a blank line below the **type** declaration.

Refer to the section Configuring Blank Lines for options regarding comments.

Violation

```
type state_machine is (idle, write, read, done);
signal sm : state_machine;
```

Fix

```
type state_machine is (idle, write, read, done);

signal sm : state_machine;
```

15.43.12 type_012

This rule checks the indent of record elements in record type declarations.

Violation

```
type interface is record
  data : std_logic_vector(31 downto 0);
chip_select : std_logic;
    wr_en : std_logic;
end record;
```

Fix

```
type interface is record
  data : std_logic_vector(31 downto 0);
  chip_select : std_logic;
  wr_en : std_logic;
end record;
```

15.43.13 type_013

This rule checks the **is** keyword in type definitions has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
type interface IS record
type interface Is record
type interface is record
```

Fix

```
type interface is record
type interface is record
type interface is record
```

15.43.14 type_014

This rule checks for consistent capitalization of type names.

Violation

```
type state_machine is (idle, write, read, done);

signal sm : State_Machine;
```

Fix

```
type state_machine is (idle, write, read, done);

signal sm : state_machine;
```

15.43.15 type_015

This rule checks for valid prefixes in user defined type identifiers. The default new type prefix is *t_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
type my_type is range -5 to 5 ;
```

Fix

```
type t_my_type is range -5 to 5 ;
```

15.43.16 type_016

This rule checks the indent of the closing parenthesis on multiline types.

Violation

```
architecture rtl of fifo is

  type state_machine is (
    idle, write, read, done
    );

begin
```

Fix

```
architecture rtl of fifo is

  type state_machine is (
    idle, write, read, done
  );

begin
```

15.43.17 type_400

This rule checks the colons are in the same column for all elements in the block declarative part.

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
type t_some_record is record
  element_1 : natural;
  some_other_element : natural;
  yet_another_element : natural;
end record;
```

Fix

```
type t_some_record is record
  element_1           : natural;
  some_other_element  : natural;
  yet_another_element : natural;
end record;
```

15.43.18 type_600

This rule checks for valid suffixes in user defined type identifiers. The default new type suffix is *_t*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
type my_type is range -5 to 5 ;
```

Fix

```
type my_type_t is range -5 to 5 ;
```

15.44 Variable Rules

15.44.1 variable_001

This rule checks the indent of variable declarations.

Violation

```
proc : process () is

variable count : integer;
      variable counter : integer;

begin
```

Fix

```
proc : process () is

  variable count : integer;
  variable counter : integer;

begin
```

15.44.2 variable_002

This rule checks the **variable** keyword has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
VARIABLE count : integer;
```

Fix

```
variable count : integer;
```

15.44.3 variable_003

This rule was depricated and replaced with rules:

- function_015
- package_019
- procedure_010
- architecture_029

15.44.4 variable_004

This rule checks the variable name has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
variable COUNT : integer;
```

Fix

```
variable count : integer;
```

15.44.5 variable_005

This rule checks there is a single space after the colon.

Violation

```
variable count   :integer;
variable counter :     integer;
```

Fix

```
variable count   : integer;
variable counter : integer;
```

15.44.6 variable_006

This rule checks for at least a single space before the colon.

Violation

```
variable count: integer;
variable counter : integer;
```

Fix

```
variable count : integer;
variable counter : integer;
```

15.44.7 variable_007

This rule checks for default assignments in variable declarations.

Violation

```
variable count : integer := 32;
```

Fix

```
variable count : integer;
```

15.44.8 variable_010

This rule checks the variable type has proper case.

Refer to the section Configuring Uppercase and Lowercase Rules for information on changing the default case.

Violation

```
variable count : INTEGER;
```

Fix

```
variable count : integer;
```

15.44.9 variable_011

This rule checks for consistent capitalization of variable names.

Violation

```
architecture rtl of entity1 is

  shared variable var1 : std_logic;
  shared variable var2 : std_logic;

begin

  proc_name : process () is

```

(continues on next page)

(continued from previous page)

```
    variable var3 : std_logic;
    variable var4 : std_logic;

  begin

    Var1 <= '0';

    if (VAR2 = '0') then
      vaR3 <= '1';
    elsif (var2 = '1') then
      VAR4 <= '0';
    end if;

  end process proc_name;

end architecture rtl;
```

Fix

```
proc_name : process () is

  variable var1 : std_logic;
  variable var2 : std_logic;
  variable var3 : std_logic;
  variable var4 : std_logic;

begin

  var1 <= '0';

  if (var2 = '0') then
    var3 <= '1';
  elsif (var2 = '1') then
    var4 <= '0';
  end if;

end process proc_name;
```

15.44.10 variable_012

This rule checks for valid prefixes on variable identifiers. The default variable prefix is *v_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed prefixes.

Violation

```
variable my_var : natural;
```

Fix

```
variable v_my_var : natural;
```

15.44.11 variable_600

This rule checks for valid suffix on variable identifiers. The default variable suffix is *v_*.

Refer to the section Configuring Prefix and Suffix Rules for information on changing the allowed suffixes.

Violation

```
variable my_var : natural;
```

Fix

```
variable my_var_v : natural;
```

15.45 Variable Assignment Rules

15.45.1 variable_assignment_001

This rule checks the indent of a variable assignment.

Violation

```
proc : process () is
begin

    counter := 0;
count := counter + 1;
```

Fix

```
proc : process () is
begin

  counter := 0;
  count   := counter + 1;
```

15.45.2 variable_assignment_002

This rule checks for a single space after the assignment.

Violation

```
counter :=0;
count   :=     counter + 1;
```

Fix

```
counter := 0;
count   := counter + 1;
```

15.45.3 variable_assignment_003

This rule checks for at least a single space before the assignment.

Violation

```
counter:= 0;
count := counter + 1;
```

Fix

```
counter := 0;
count := counter + 1;
```

15.45.4 variable_assignment_004

This rule checks the alignment of multiline variable assignments.

Violation

```
counter := 1 + 4 + 10 + 25 +
     30 + 35;
```

Fix

```
counter := 1 + 4 + 10 + 25 +
           30 + 35;
```

15.45.5 variable_assignment_005

This rule checks the alignment of **:=** operators over multiple lines.

Following extra configurations are supported:

- `if_control_statements_end_group`,
- `case_control_statements_end_group`,

Refer to the section Configuring Keyword Alignment Rules for information on changing the configurations.

Violation

```
counter := 0;
count  := counter + 1;
```

Fix

```
counter := 0;
count   := counter + 1;
```

15.45.6 variable_assignment_006

This rule checks for comments in multiline variable assignments.

Violation

```
counter := 1 + 4 + 10 + 25 +
           -- Add in more stuff
           30 + 35;
```

Fix

```
counter := 1 + 4 + 10 + 25 +
           30 + 35;
```

15.46 Wait Rules

15.46.1 wait_001

This rule checks for indentation of the **wait** keyword. Proper indentation enhances comprehension.

Violation

```
begin

    wait for 10ns;
 wait on a,b;
       wait until a = '0';
```

Fix

```
begin

  wait for 10ns;
  wait on a,b;
  wait until a = '0';
```

15.47 When Rules

These rules cover the usage of **when** keywords in sequential and concurrent statements.

15.47.1 when_001

This rule checks the **else** keyword is not at the beginning of a line. The else should be at the end of the preceeding line.

Violation

```
wr_en <= '1' when a = '1' -- This is comment
         else '0' when b = '0'
         else c when d = '1'
         else f;
```

Fix

```
wr_en <= '1' when a = '1' else -- This is a comment
         '0' when b = '0' else
         c when d = '1' else
         f;
```

15.48 While Loop Rules

15.48.1 while_loop_001

This rule checks for indentation of the **while** keyword. Proper indentation enhances comprehension.

Violation

```
begin

while (temp /= 0) loop
    temp := temp/2;
  end loop;
```

Fix

```
begin

  while (temp /= 0) loop
    temp := temp/2;
  end loop;
```

15.48.2 while_loop_002

This rule checks for indentation of the **end loop** keywords. The **end loop** must line up with the **while** keyword. Proper indentation enhances comprehension.

Violation

```
begin

  while (temp /= 0) loop
    temp := temp/2;
      end loop;
```

Fix

```
begin

  while (temp /= 0) loop
    temp := temp/2;
  end loop;
```

15.49 Whitespace Rules

15.49.1 whitespace_001

This rule has been depricated.

VSG strips trailing spaces when a file is read in.

15.49.2 whitespace_002

This rule has been depricated.

VSG changes tabs to spaces when a file is read in.

15.49.3 whitespace_003

This rule checks for spaces before semicolons.

Violation

```
wr_en : in    std_logic      ;
```

Fix

```
wr_en : in    std_logic;
```

15.49.4 whitespace_004

This rule checks for spaces before commas.

Violation

```
wr_en => wr_en    ,
rd_en => rd_en,
```

Fix

```
wr_en => wr_en,
rd_en => rd_en,
```

15.49.5 whitespace_005

This rule checks for spaces after an open parenthesis.

Note: Spaces before numbers are allowed.

Violation

```
signal data        : std_logic_vector(31 downto 0);
signal byte_enable : std_logic_vector( 3 downto 0);
signal width       : std_logic_vector(   g_width - 1 downto 0);
```

Fix

```
signal data        : std_logic_vector(31 downto 0);
signal byte_enable : std_logic_vector( 3 downto 0);
signal width       : std_logic_vector(g_width - 1 downto 0);
```

15.49.6 whitespace_006

This rule checks for spaces before a close parenthesis.

Violation

```
signal data        : std_logic_vector(31 downto 0    );
signal byte_enable : std_logic_vector( 3 downto 0 );
signal width       : std_logic_vector(g_width - 1 downto 0);
```

Fix

```
signal data        : std_logic_vector(31 downto 0);
signal byte_enable : std_logic_vector( 3 downto 0);
signal width       : std_logic_vector(g_width - 1 downto 0);
```

15.49.7 whitespace_007

This rule checks for spaces after a comma.

Violation

```
proc : process (wr_en,rd_en,overflow) is
```

Fix

```
proc : process (wr_en, rd_en, overflow) is
```

15.49.8 whitespace_008

This rule checks for spaces after the **std_logic_vector** keyword.

Violation

```
signal data    : std_logic_vector (7 downto 0);
signal counter : std_logic_vector    (7 downto 0);
```

Fix

```
signal data    : std_logic_vector(7 downto 0);
signal counter : std_logic_vector(7 downto 0);
```

15.49.9 whitespace_010

This rule checks for spaces before and after the concate (&) operator.

Violation

```
a <= b&c;
```

Fix

```
a <= b & c;
```

15.49.10 whitespace_011

This rule checks for at least a single space before and after math operators +, -, /, * and **.

Violation

```
a <= b+c;
a <= b-c;
a <= b/c;
a <= b*c;
a <= b**c;
a <= (b+c)-(d-e);
```

Fix

```
a <= b + c;
a <= b - c;
a <= b / c;
a <= b * c;
a <= b ** c;
a <= (b + c) - (d - e);
```

15.49.11 whitespace_012

This rule enforces a maximum number of consecutive blank lines.

Violation

```
a <= b;


c <= d;
```

Fix

```
a <= b;

c <= d;
```

Note: The default is set to 1. This can be changed by setting the *numBlankLines* attribute to another number.

```
{
    "rule":{
        "whitespace_012":{
            "numBlankLines":3
        }
    }
}
```

15.49.12 whitespace_013

This rule checks for at least a single space before and after logical operators.

Violation

```
if (a = '1')and(b = '0')
if (a = '0')or (b = '1')
```

Fix

```
if (a = '1') and (b = '0')
if (a = '0') or (b = '1')
```

15.50 With Rules

15.50.1 with_001

This rule checks for **with** statements.

Violation

```
with buttons select
```

Fix

Refactor **with** statement into a process.

CHAPTER 16

Contributing

I welcome any contributions to this project. No matter how small or large.

There are several ways to contribute:

1. Bug reports
2. Code base improvements
3. Feature requests
4. Pull requests

16.1 Bug Reports

I used code from open cores to develop VSG. It provided many different coding styles to process. There are bound to be some corner cases or incorrect assumptions in the code. If you run into anything that is not handled correctly, please submit an issue. When creating the issue, use the **bug** label to highlight it. Fixing bugs is prioritized over feature enhancements.

16.2 Code Base Improvements

VSG started out to solve a problem and learn how to code in Python. The learning part is still on going, and I am sure the code base could be improved. I run the code through *Codacy* and *Code Climate*, and they are very helpful. However, I would appreciate any suggestions to improve the code base.

Create an issue and use the **refactor** label for any code which could be improved.

16.3 Feature Requests

Let me know if there is anything I could add to VSG easier to use. The following features were not in my original concept of VSG.

- fix
- fix_phase
- output_format
- backup

Fix is probably the most important feature of VSG. I added it when someone said it would be nice if VSG just fixed the problems it found. There may be other important features, I just have not thought of them yet.

If you have an idea for a new feature, create an issue with the **enhancement** label.

16.4 Pull Requests

Pull requests are always welcome. I am trying to follow a Test Driven Development (TDD) process. Currently there are over 1000 tests. If you do add a new feature or fix a bug, I would appreciate a new or updated test to go along with the change.

I use *Travis CI* to run all the tests. I also use *Codacy* and *Code Climate* to check for code style issues. I use *Codcov* to check the code coverage of the tests.

Travis CI will run these tools when a pull request is made. The results will be available on the pull request Github page.

16.5 Running Tests

Before submitting a pull request, you can run the existing tests locally. These are the same tests Travis CI will run.

To run the tests issue the following command when using python 2.7:

```
python -m unittest discover
```

To run the tests using python 3 use the following command:

```
python -m unittest
```

After issuing the command the tests will be executed.

```
vhdl-style-guide$ python -m unittest discover
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
```

(continues on next page)

(continued from previous page)

```
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
.................................................
...........................................
-------------------------------------------------
Ran 1170 tests in 6.424s

OK
```

CHAPTER 17

Release Notes

Release notes are maintained with the project on github.

https://github.com/jeremiah-c-leary/vhdl-style-guide/releases