

Leda

VRSL Reference Guide

Version 2006.06
June 2006



Comments?
E-mail your comments about this manual to
leda-support@synopsys.com.

SYNOPSYS[®]

Copyright Notice and Proprietary Information

Copyright © 2005 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSPICE, Hypermodel, iN-Phase, in-Sync, Leda, MAST, Meta, Meta-Software, ModelAccess, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SiVL, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAI, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert Plus, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, Discovery, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDANavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA Express, Frame Compiler, Galaxy, Gattran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JvXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

All other product or company names may be trademarks of their respective owners.

Contents

Preface	11
About This Manual	11
Related Documents	11
Manual Overview	11
Typographical and Symbol Conventions	12
Getting Leda Help	13
The Synopsys Web Site	13
Chapter 1	
VHDL Rule Specification Language	15
Introduction	15
What is VRSL?	16
VRSL Templates and Attributes	16
VRSL Templatesets and Rulesets	17
VRSL Rules	17
VRSL Commands	18
VRSL Attributes	18
VRSL Context	21
VRSL Message	21
VRSL Severity	22
VRSL Examples	22
A First Example	22
Conditional Commands	24
ANDing Template Matching	25
Aggregate Attributes	26
Parameterizing Rules Using Regular Expressions	28
Constrain Max/Min Attributes to Predefined Values	30
Linking to HTML Files	31
Parameterizing Error Messages	32
Limitations on use of <%item>	33
Formal Definition of VRSL	36
Templateset	36
Ruleset	36
Declarative Part	37
Template No Command	37

Template Force Command	37
Template Limit Command	38
Template Set Command	39
Template Max or Min Command	39
Command Part	40
No Command	40
Force Command	41
Limit Command	42
Set Command	42
Max or Min Command	43
VRSL Primitives	44
Literal_Type	44
Evaluation_Periods	45
Error_Status	45
Recursion_Type	45
Edge_Type	46
VRSL Classes	47
CONCURRENT_STATEMENT	48
DECLARATIVE_ITEM	48
DISCRETE_RANGE	49
DRIVEN_OBJECT	49
EXPRESSION	49
ID	50
NAME	50
NAMED_ENTITY	50
OBJECT_ITEM	51
REGION_PART	51
SEQUENTIAL_STATEMENT	52
STATEMENT	53
TARGET	54
TYPE_MARK	54
TYPE_DEFINITION	54
Chapter 2	
Templates and Attributes	55
Introduction	55
Chip-Level Templates and Attributes	55
Connectivity Path Template	56
Data Signal Template	58
Design Template	58

Flipflop Template	60
Latch Template	62
Logic Cost Template	63
Test Signal Template	65
Block-Level Templates and Attributes	66
Access Type Definition Template	70
Aggregate Template	70
Alias Declaration Template	71
Allocator Template	72
Architecture Body Template	73
Assertion Statement Template	77
Association Element Template	78
Association List Template	79
Asynchronous Initialization Template	80
Attribute Declaration Template	81
Attribute Name Template	82
Attribute Specification Template	83
Binary Operation Template	85
Binding Indication Template	86
Block Configuration Template	87
Block Specification Template	88
Block Statement Template	89
Case Statement Template	91
Clock Template	93
Comment Template	94
Component Configuration Template	95
Component Declaration Template	95
Component Instantiation Statement Template	97
Component Specification Template	98
Concurrent Assertion Statement Template	99
Concurrent Procedure Call Statement Template	100
Conditional Signal Assignment Template	101
Conditional Waveforms Template	102
Configuration Declaration Template	104
Configuration Specification Template	106
Constant Declaration Template	106
Constrained Array Definition Template	107
Design Unit Template	108
Disconnection Specification Template	109
Element Association Template	109

Element Declaration Template	110
Entity Declaration Template	111
Enumeration Type Definition Template	115
Exit Statement Template	116
Expression Template	117
FSM Template	118
File Declaration Template	119
File Layout Template	120
File Type Definition Template	121
Flipflop Template	121
Floating Type Definition Template	122
For Loop Statement Template	123
Formal Parameter Template	125
Function Call Template	126
Generate Statement Template	128
Generic Declaration Template	130
Group Declarations	131
Group Template Declarations	131
Header Comment Template	132
Identifier Template	133
If Statement Template	134
Indexed Name Template	136
Integer Type Definition Template	137
Interface File Declaration Template	138
Interface Variable Declaration Template	139
Latch Template	140
Literal Template	141
Loop Statement Template	143
Next Statement Template	145
Package Body Template	146
Package Declaration Template	148
Physical Type Definition Template	150
Port Declaration Template	151
Procedure Call Statement Template	154
Process Statement Template	155
Range Template	159
Record Type Definition Template	161
Report Statement Template	161
Return Statement Template	162
Selected Name Template	163

Selected Signal Assignment Template	164
Selected Waveforms Template	165
Shared Variable Declaration Template	166
Signal Assignment Statement Template	167
Signal Declaration Template	168
Simple Name Template	170
Slice Name Template	171
Statement Format Template	172
Subprogram Body Template	173
Subprogram Declaration Template	178
Subtype Declaration Template	180
Subtype Indication Template	181
Synchronous Initialization Template	182
Type Declaration Template	183
Unconstrained Array Definition Template	184
Use Clause Template	185
Variable Assignment Statement Template	185
Variable Declaration Template	186
Wait Statement Template	187
Waveform Element Template	188
Waveform Template	189
While Loop Statement Template	189
Appendix A	
Template x Attribute SpecDex	193
About the SpecDex	193
Appendix B	
Attribute x Template SpecDex	233
About the SpecDex	233
Index	287

Tables

Table 1:	Documentation Conventions	12
Table 2:	VRSL Rule Components	17
Table 3:	VRSL Commands	18
Table 4:	Types of VRSL Attributes	18
Table 5:	Leda Error Message Parameters	32
Table 6:	connectivity_path Template	56
Table 7:	data_signal Template	58
Table 8:	design Template	58
Table 9:	flipflop Template	60
Table 10:	latch Template	62
Table 11:	logic_cost Template	63
Table 12:	test_signal Template	65
Table 13:	access_type_definition Template	70
Table 14:	aggregate Template	70
Table 15:	alias_declaration Template	71
Table 16:	allocator Template	72
Table 17:	architecture_body Template	73
Table 18:	assertion_statement Template	77
Table 19:	association_element Template	78
Table 20:	association_list Template	79
Table 21:	asynchronous_initialization Template	80
Table 22:	attribute_declaration Template	81
Table 23:	attribute_name Template	82
Table 24:	attribute_specification Template	83
Table 25:	binary_operation Template	85
Table 26:	binding_indication Template	86
Table 27:	block_configuration Template	87
Table 28:	block_specification Template	88
Table 29:	block_statement Template	89
Table 30:	case_statement Template	91
Table 31:	clock Template	93
Table 32:	comment Template	94
Table 33:	component_configuration Template	95
Table 34:	component_declaration Template	95
Table 35:	component_instantiation_statement Template	97
Table 36:	component_specification Template	98

Table 37: concurrent_assertion_statement Template	99
Table 38: concurrent_procedure_call_statement Template	100
Table 39: conditional_signal_assignment Template	101
Table 40: conditional_waveforms Template	103
Table 41: configuration_declaration Template	104
Table 42: configuration_specification Template	106
Table 43: constant_declaration Template	106
Table 44: constrained_array_definition Template	107
Table 45: design_unit Template	108
Table 46: disconnection_specification Template	109
Table 47: element_association Template	109
Table 48: element_declaration Template	110
Table 49: entity_declaration Template	112
Table 50: enumeration_type Template	115
Table 51: exit_statement Template	116
Table 52: expression Template	117
Table 53: fsm Template	118
Table 54: file_declaration Template	119
Table 55: file_layout Template	120
Table 56: file_type_definition Template	121
Table 57: flipflop Template	121
Table 58: floating_type_definition Template	122
Table 59: for_loop_statement Template	123
Table 60: formal_parameter Template	125
Table 61: function_call Template	126
Table 62: generate_statement Template	128
Table 63: generic_declaration Template	130
Table 64: header_comment Template	132
Table 65: identifier Template	133
Table 66: if_statement Template	134
Table 67: indexed_name Template	136
Table 68: integer_type_definition Template	137
Table 69: interface_file_declaration Template	138
Table 70: interface_variable_declaration Template	139
Table 71: latch Template	140
Table 72: literal Template	141
Table 73: loop_statement Template	143
Table 74: next_statement Template	145
Table 75: package_body Template	146
Table 76: package_declaration Template	148

Table 77: physical_type_definition Template	150
Table 78: port_declaration Template	151
Table 79: procedure_call_statement Template	154
Table 80: process_statement Template	155
Table 81: range Template	159
Table 82: record_type_definition Template	161
Table 83: report_statement Template	161
Table 84: return_statement Template	162
Table 85: selected_name Template	163
Table 86: selected_signal_assignment Template	164
Table 87: selected_waveforms Template	165
Table 88: shared_variable_declaration Template	166
Table 89: signal_assignment_statement Template	167
Table 90: signal_declaration Template	168
Table 91: simple_name Template	170
Table 92: slice_name Template	171
Table 93: statement_format Template	172
Table 94: subprogram_body Template	174
Table 95: subprogram_declaration Template	178
Table 96: subtype_declaration Template	180
Table 97: subtype_indication Template	181
Table 98: synchronous_initialization Template	182
Table 99: type_declaration Template	183
Table 100: unconstrained_array_definition Template	184
Table 101: use_clause Template	185
Table 102: variable_assignment_statement Template	185
Table 103: variable_declaration Template	186
Table 104: wait_statement Template	187
Table 105: waveform_element Template	188
Table 106: waveform Template	189
Table 107: while_loop_statement Template	189

Preface

About This Manual

This manual is designed for engineers who want to develop rules in the VHDL Rule Specification Language (VRSL). If you are unfamiliar with VRSL, you should read and work the examples in the *Leda Rule Specifier Tutorial* before consulting this book. This manual is intended for use by design and quality assurance engineers who are already familiar with VRSL and VHDL.

Related Documents

This manual is part of the Leda documentation set. To see a complete listing, refer to the *Leda Document Navigator*.

Manual Overview

This manual contains the following chapters and appendixes:

Preface	Describes the manual and lists the typographical conventions and symbols used in it; tells how to get technical assistance.
Chapter 1 VHDL Rule Specification Language	Overview and detailed description of the VHDL Rule Specification Language (VRSL).
Chapter 2 Templates and Attributes	An API-like reference for all of the VRSL templates and attributes that you use to write rules.
Appendix A Template x Attribute SpecDex	Hyperlinked lookup tool for finding all attributes associated with each template.

[Appendix B](#)
[Attribute x Template SpecDex](#)

Hyperlinked lookup tool for finding all templates associated with each attribute.

Typographical and Symbol Conventions

The following conventions are used throughout this document:

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). <pre>% cd \$LMC_HOME/hd1</pre>
Monospace	System-generated text (prompts, messages, files, reports). <pre>No Mismatches: 66 Vectors processed: 66 Possible"</pre>
<i>Italic or Italic</i>	Variables for which you supply a specific value. As a command line example: <pre>% setenv LMC_HOME prod_dir</pre> In body text: In the previous example, <i>prod_dir</i> is the directory where your product must be installed.
(Vertical rule)	Choice among alternatives, as in the following syntax example: <pre>-effort_level low medium high</pre>
[] (Square brackets)	Enclose optional parameters: <pre>pin1 [pin2 ... pinN]</pre> In this example, you must enter at least one pin name (<i>pin1</i>), but others are optional (<i>[pin2 ... pinN]</i>).
TopMenu > SubMenu	Pulldown menu paths, such as: File > Save As ...

Getting Leda Help

For help with Leda, send a detailed explanation of the problem, including contact information, to leda-support@synopsys.com.

The Synopsys Web Site

General information about Synopsys and its products is available at this URL:

<http://www.synopsys.com>

1

VHDL Rule Specification Language

Introduction

This chapter provides reference information for the VHDL Rule Specification Language (VRSL). You use VRSL to write custom coding rules when there isn't a prepackaged rule that meets your design team's needs.



Note

For hardware rules, you use C or Tcl instead to write custom rules and integrate them into the Leda environment using VerSL wrappers. For details on how to do this, see the [Leda Tcl Interface Guide](#) or the [Leda C Interface Guide](#).

The information in this chapter is organized in the following major sections:

- [“What is VRSL?” on page 16](#)
- [“VRSL Templates and Attributes” on page 16](#)
- [“VRSL Templatesets and Rulesets” on page 17](#)
- [“VRSL Rules” on page 17](#)
- [“VRSL Examples” on page 22](#)
- [“Parameterizing Error Messages” on page 32](#)
- [“Formal Definition of VRSL” on page 36](#)
- [“VRSL Primitives” on page 44](#)
- [“VRSL Classes” on page 47](#)

What is VRSL?

VRSL is a macro-based language that you use to write rules that check VHDL code for errors or anomalies. You write rules using prebuilt templates and attributes that match certain kinds of VHDL code, and a simple set of commands. There are six VRSL commands: force, no, limit, set, max, and min. Each command has a precise syntax with allowed keywords.

Most of the terminology used for writing rules comes from the *IEEE Standard 1076-1993 VHDL Language Reference Manual* (LRM). The LRM is the basis upon which this manual was created. Most of the VRSL keywords correspond to VHDL grammar rules and clauses, and can be found in the LRM. However, to avoid excessively long rule chains for VHDL structures such as expressions, VRSL does not always follow the LRM grammar. These deviations are noted in this manual where they apply.

VRSL's extra keywords allow you to easily define application-specific rules. For example, for synthesis it may be necessary to distinguish between a combinatorial and a sequential process or to limit the number of clocks in a process.



Note

To distinguish between VHDL rules defined in the LRM and rules for a given coding style, the former are referred to as `VHDL_rules` in this manual.

VRSL Templates and Attributes

Templates and attributes are the building blocks that you combine with VRSL commands to write rules.

A template defines a model of how the VHDL code should appear. Templates are basic elements of VRSL code that you use to build rules or even other templates. Templates are all prepackaged (VRSL primary template or VRSL secondary template). You can assign any string to be (template *my_template* is *template*) where *template* is one of the prepackaged templates and define its focus using VRSL commands, but you cannot create new templates or attributes yourself.

Each template has a set of attributes or characteristics of VHDL code that you can use with it. When you define a template to model the VHDL code you want to constrain, you select one or more attributes from this set and use VRSL commands like force, no, or limit to precisely define that model or template. Then you write a rule that calls that template and constrains the code that the template matches.

VRSL Templatesets and Rulesets

VRSL contains two types of high-level semantic units: templatesets and rulesets. A templateset is like a VHDL package. It contains a set of template declarations. No commands are allowed in templatesets, but they can contain other templateset units. A ruleset is similar to a VHDL architecture. Rulesets can contain template declarations, commands, and other templateset units.

VRSL Rules

A VRSL rule can contain five parts, most often in this order:

`command attribute context message severity`

[Table 2](#) describes each of these components.

Table 2: VRSL Rule Components

Rule Component	Description	Type
command	Keyword that specifies how to constrain a VHDL item.	Required
attribute	Keyword that specifies the VHDL item to be constrained.	Required
context	Keyword that refines the focus of where the VHDL item is to be constrained. Note: You don't need to specify a context when you use a primary template for a rule, but you must provide a context when you use a secondary template for a rule.	Required when using secondary templates
message	Text to be displayed in the Checker's Error Viewer when the rule is violated.	Optional
severity	The severity level for violation of this rule (note, warning, error, or fatal)	Optional

The following sections provide more information about each part of a VRSL rule:

- [“VRSL Commands” on page 18](#)
- [“VRSL Attributes” on page 18](#)
- [“VRSL Context” on page 21](#)
- [“VRSL Message” on page 21](#)
- [“VRSL Severity” on page 22](#)

VRSL Commands

[Table 3](#) explains how each of the VRSL commands works.

Table 3: VRSL Commands

Command	Use
no	The specified VHDL_rule or clause must not be present. If it is present, the Checker flags a rule violation.
force	The specified VHDL_rule or clause must be present. If it is not present, the Checker flags a rule violation.
limit	The specified VHDL_rule or clause must match at least one of the specified set of templates. If none of the specified templates match, the Checker flags a rule violation. The limit command is the only type of VRSL command that you can use with conditional logic (for example, if ..., then ...).
set	The specified VHDL_rule must match the specified fixed value. If it does not match, the Checker flags a rule violation.
max	The maximum number of times that the specified VHDL_rule may appear. If the maximum number is exceeded, the Checker flags a rule violation.
min	The minimum number of times that the specified VHDL_rule must appear. If the minimum number is not met, the Checker flags a rule violation.

VRSL Attributes

An attribute represents the VHDL item that you want to constrain. There are several hundred attributes in VRSL, and you can use them in a variety of ways, depending on their type. There are five different types of attributes. Each type is restricted for use with certain VRSL commands, as explained in [Table 4](#).

Table 4: Types of VRSL Attributes

Type	Description	Use Only with These Commands
template_kind	These can be attributes of a given template and also function as templates themselves, with their own attributes. How you use this type of attribute depends on how tightly you want to focus your rule. For a wide view, use the attribute as part of another template. For a close-in view, use the attribute as its own template. For example, template_kind attributes include all statements and declarations.	<ul style="list-style-type: none"> • limit • no • force

Table 4: Types of VRSL Attributes (Continued)

Type	Description	Use Only with These Commands
local_attribute	<p>These attributes can belong to one or more templates, but they cannot themselves function as templates. For example, you can use the <code>downto</code> attribute to control the use of the <code>downto</code> keyword.</p>	<ul style="list-style-type: none"> • no • force
set_attribute	<p>These attributes have precise meanings that are represented by enumerated types or strings. For example, you can use the <code>evaluation_time</code> attribute of expression templates to indicate if the expression should be locally or globally static. You specify this using a built-in enumerated type as follows:</p> <pre style="margin-left: 40px;">set evaluation_time to locally_static_evaluation</pre>	<ul style="list-style-type: none"> • set
max_min_attribute	<p>These attributes are similar to set attributes, but you use them strictly for specifying bounds on certain attributes. For example, you can use the <code>dimension</code> attribute, (which is a <code>max_min</code> type) to specify the maximum or minimum number of characters in an array.</p>	<ul style="list-style-type: none"> • max • min
aggregate_attribute	<p>To impose constraints on different occurrences of the same attribute, you use <code>aggregate_attributes</code>. These attributes all have the “_s” suffix. For example, the <code>condition_s</code> attribute of the <code>if_statement</code> template is an <code>aggregate_attribute</code>.</p>	<ul style="list-style-type: none"> • aggregate limit • no • force

Attribute Examples

Following are some example VRSL rules that show the attributes in **bold** for easy identification:

No Wait Statements

```
no wait_statement
```

In this example, if a wait statement appears anywhere in the design entity, the rule is violated.

No Rising Edge in Condition of Wait Statements

```
no RISING_EDGE_1 in condition of wait_statement
```

In this example, if an expression in the condition clause of a wait_statement matches the template **RISING_EDGE**, the rule is violated. Note that, in most cases, the attribute comes immediately after the command. However, this example, where you specify that something should not match, is an exception to this general rule.

Limit Condition to Rising Edge

```
limit condition in wait_statement to RISING_EDGE
```

In this example, if the expression in the condition of each wait statement does not match the template **RISING_EDGE**, the rule is violated.

Max Dimension Count in Constrained Array

```
max dimension_count in constrained_array_definition is 1
```

In this example, if the number of dimensions in a constrained array declaration is greater than 1, the rule is violated.

Attribute-Related Compilation Errors

Because VRSL constrains the kinds of attributes that you can use with the different commands, you can get a compilation error if you use an attribute with a command for which it is not allowed. For example, the following VRSL rule:

```
limit dimension_count in constrained_array_definition to 1 -- Invalid rule
```

produces a compilation error because you cannot use the `dimension_count` attribute with the limit command.

VRSL Context

The context specifies where the VRSL command applies. A context is usually a template or attribute, depending on how broad or narrow you want the application of your rule to be. You don't need to specify a context when you use a primary template for a rule, but you must provide a context when you use a secondary template for a rule.

Following are some example VRSL rules that show the context in **bold** for easy identification:

```
no wait_statement in subprogram_body
no RISING_EDGE_1 in condition of wait_statement
limit condition in wait_statement to RISING_EDGE
max dimension_count in constrained_array_definition is 1
```

Notice how the context is usually preceded by the “in” keyword. This is the general rule. The exception is shown in the second example, where you specify something you don't want to be present. In this case, you precede the context with the “of” keyword.

Note that some contexts are not valid for a given attribute. For example:

```
no wait_statement in package_declaration -- Invalid rule
```

produces an error message because wait statements cannot appear in package declarations. If you instead specify the context as “all”:

```
no wait_statement in all-- Valid rule
```

Leda checks all appropriate contexts.

VRSL Message

The message part of a VRSL rule is defined as:

```
message ::= message_string [[html-doc]html_app]]
```

The message that you specify when writing a rule later appears in the Error Viewer when that rule is violated. Note that a message only appears if the severity level for that rule is high enough to get past the filtering options that you set.

You use the `html_doc` and `html_app` clauses to specify links to HTML-based help documentation for a rule. For more information, see [“Linking to HTML Files” on page 31](#).

VRSL Severity

Use the severity portion of a VRSL rule to inform users about the relative seriousness of the infraction when this rule is violated. The possible values are note, warning, error, and fatal. A fatal error does not cause the Leda tool to crash, but is an indication that a violation of this kind in your VHDL code may cause serious problems for downstream tools in the design and verification flow.

If you do not specify a severity for a rule, Leda use the default value, which is note.

VRSL Examples

The following sections explain basic features of the VRSL language, using examples to demonstrate how to use them:

- [“A First Example” on page 22](#)
- [“Conditional Commands” on page 24](#)
- [“ANDing Template Matching” on page 25](#)
- [“Aggregate Attributes” on page 26](#)
- [“Parameterizing Rules Using Regular Expressions” on page 28](#)
- [“Constrain Max/Min Attributes to Predefined Values” on page 30](#)
- [“Linking to HTML Files” on page 31](#)

A First Example

This first example uses the VHDL `wait_statement`. The VHDL_rule for the `wait_statement` is as follows (LRM §8.1):

```
wait_statement ::=
    [ label : ] wait [ sensitivity_clause ] [ condition_clause ]
    [timeout_clause ] ;
sensitivity_clause ::= on sensitivity_list
sensitivity_list ::= signal_name { , signal_name }
condition_clause ::= until condition
condition ::= boolean_expression
timeout_clause ::= for time_expression
```

Notice that the `wait_statement` is subdivided into four clauses: label, sensitivity, condition, and timeout.

**Note**

Although tokens such as `label` are not exactly clauses, throughout this document this term is used to mean any subelement of a `VHDL_rule`.

Leda must be able to refine all of these clauses with different degrees of granularity. For example:

- Limit condition clauses to expressions modeling rising edges
- Forbid timeout clauses

For this first rule, assume that an expression template called `RISING_EDGE_1` has been declared in the templateset `CLOCK_EDGES` to model rising-edge expressions.

```
ruleset SYNTHESIS is
    use templateset CLOCK_EDGES
    limit condition in wait_statement to RISING_EDGE_1
        message "SYN_8.1-3 Illegal condition in wait statement"
        severity error
end ruleset
```

To ensure that the condition clause is always present and that it matches the `RISING_EDGE` expression template, you could write the following commands:

```
force condition in wait_statement
    message "Condition must be present in wait statement"
    severity error
limit condition in wait_statement to RISING_EDGE_1
    message "SYN_8.1-3 Illegal condition in wait statement"
    severity error
```

Similarly, the second rule (forbid timeout clauses) is relevant to all wait statements. You can describe this in the command part as follows:

```
no timeout in wait_statement
    message "SYN8.1-4 Timeout clauses are illegal"
    severity error
```

You can define the same rules in a `wait_statement` template as follows:

```
template WAIT is wait_statement
    limit condition to RISING_EDGE_1
    no timeout
end
```

You could then use this template to restrict the use of wait statements in different contexts. For example:

```
limit wait_statement in process_statement to WAIT
    message "Illegal wait statement in process"
    severity error
no wait_statement in subprogram_body
    message "Wait statements are illegal in subprograms"
    severity error
```

Conditional Commands

In some cases, depending on the outcome of one rule, you may need to apply a different set of rules. To do that you can use conditional logic (for example, if ..., then ,,) with VRSL limit commands. Note that limit commands are the only VRSL commands that you can use with conditional logic.

Suppose you want the following rules:

- Architectures may only have names “RTL” or “STRUCT”
- Component instantiations are not allowed in RTL architectures
- Process statements are not allowed in STRUCT architectures

To limit architecture body identifiers, you can declare the following templates:

```
template RTL_ARCH is architecture_body
    limit identifier to "RTL"
end
template STRUCT_ARCH is architecture_body
    limit identifier to "STRUCTURAL"
end
```

and then define the command restricting the architecture body as follows:

```
limit architecture_body in all to RTL_ARCH, STRUCT_ARCH
    message "Illegal name for architecture_body"
    severity error
```


Next, you define rules for limiting the contents of an architecture body depending on which template was matched. For the example above, you could use conditional commands as follows:

```
limit architecture_body in all to RTL_ARCH, STRUCT_ARCH
  message "Illegal name for architecture_body"
  severity error
  if RTL_ARCH then
    no component_instantiation_statement
    message "Component instantiation statements are illegal"
    severity error
  end if
  if STRUCT_ARCH then
    no process_statement
    message "Processes are illegal"
    severity error
  end if
```

So, if RTL_ARCH matches, the rule forbids component instantiation statements. If STRUCT_ARCH matches, the rule forbids process statements.

ANDing Template Matching

The previous example showed how to use the limit command to match a piece of VHDL code with one of a set of templates. Suppose, instead, that you want to match the VHDL code with all templates in a list. If you wanted to limit port map aspects so that:

- Only named associations are used, and
- No others clauses are used

you could write the rule as follows:

```
template PORT_MAP_ASSOC is association_list
  force named_association
  no others
end
limit port_map_aspect in component_instantiation_statement
  to PORT_MAP_ASSOC
  message "Illegal port map aspect"
  severity Error
```

The disadvantage of this coding style is that you have to specify a very general message with the limit rule. For more specific messages, you can use the `allof` keyword to indicate that you want to match against every template in the list. Then for each mismatch, you can specify a different message, as follows:

```

template NAMED_ASSOC is association_list
    force named_association
end
template NO_OTHERS is association_list
    no others
end

limit port_map_aspect in component_instantiation_statement
to allof
    NAMED_ASSOC message "Use named association in port maps",
    NO_OTHERS message "Others clause is illegal in port maps"
severity error

```

Aggregate Attributes

Some coding styles require that VHDL statements or clauses appear in a certain order. For example, some synthesis tools can optimize processes written according to a certain profile. Consider the following VHDL code:

```

FF_ARST: process (CLK,RST)
begin
    if RST='1' then
        Q<='0';
    elsif CLK='1' AND CLK' event then
        Q<=D;
    end if;
end process FF_ARST;

```

This represents a flip-flop with an asynchronous reset. The process contains a sensitivity list (no waits) and a single statement in its outermost region. The if statement has two conditions (no else), and these two conditions are expressions of a predefined format: the first is a high value on RST and the second is a rising edge on CLK.

Because you often need to assign different expressions to different conditions of an if statement, VRSL has the concept of aggregate attributes. These are sets of attributes that can be tested against different templates for each item. For example, the following command limits if statement conditions to those described above:

```

limit condition_s in if_statement to
    first HIGH_LEVEL, then RISING_EDGE_1
message "Illegal condition in if statement"
severity error

```

In this example, `HIGH_LEVEL` and `RISING_EDGE_1` are predefined expression templates that are located in the `CLOCK_EDGES` templateset.

Aggregate attributes always have an “_s” suffix. Other aggregate attributes include:

- `association_element_s`
- `conditional_waveforms_s`
- `element_association_s`
- `index_constraint_s`
- `selected_waveforms_s`
- `statement_profile_s`
- `waveform_element_s`

You can use aggregate attributes as ordinary attributes by assigning a single list of templates and removing the first keyword. For example, the following code means that every condition alternative is checked against both predefined templates.

```
limit condition_s in if_statement to HIGH_LEVEL, RISING_EDGE_1
```

You can also test different occurrences of the same VHDL structure against the same set of templates. Suppose in the example above you want to restrict the statements in the outermost statement region of the process to any number of variable assignment statements (corresponding to initializations) followed by the if statement. You can write this rule as follows:

```
limit statement_profile_s in process_statement to
    first {VAR_ASS_STM}, then IF1
    message "Illegal process profile"
    severity error
```

where `VAR_ASS_STM` is a previously defined template for `variable_assignment_statements` and `IF1` is an `if_statement` template limiting the `condition_s` attribute as shown above.

Putting braces around the template name, `{VAR_ASS_STM}`, means that all VHDL statements are checked against `VAR_ASS_STM` until a mismatch occurs. If the VHDL statement that causes the mismatch doesn't match `IF1`, you get an error. If it does match,

then Leda moves on to the next set of templates. For example, suppose you want every element in an association list to match with the same `association_element` template. You can write the VRSL rule for this as follows:

```
template ASSOC_ELT is association_element
    no formal_function
end
template ASSOC_LIST is association_list
    limit association_element_s to {ASSOC_ELT}
end
limit association_list to ASSOC_LIST
    message "Illegal association_list"
    severity error
```

The braces around `{ASSOC_ELT}` in this example mean that the VHDL code is not restricted. You should only use braces around a template names with the last “then” alternative.

Parameterizing Rules Using Regular Expressions

Naming conventions are not part of VHDL, but such rules frequently appear in coding standards. To support rules for naming convention rules, Leda accepts regular expressions in strings used to constrain names.



Note

In general, results you get using regular expressions with Leda are the same as you get using `grep`. Both programs use `regex (5)` to parse regular expressions. Note that Leda currently supports simple regular expressions, but not extended regular expressions of the form `\{m\}`, `\{m,\}`, or `\{m,n\}` that are supported with `egrep` or `grep -E`. Also, Leda uses the GNU version of `regex`, which differs slightly from the UNIX version (for details see the man pages).

For example, suppose you want to constrain the labels of component instantiation statements so that they have one of the following formats:

- `<instantiated entity name>0` or
- `<instantiated entity name>_<index>`
- `<instantiated entity name>_<more_info>_<index>`

where `<index>` is an integer value and `<more_info>` is a user-defined string.

The following VRSL command correctly tests the instantiation statements:

```
limit label in component_instantiation_statement to
    "^<entity>0$",
    "^<entity>_[0-9]*$",
    "^<entity>_[.]*_ [0-9]*$"
```

where <entity> is a predefined parameter that is replaced by the name of the enclosing entity.

You can define variable parameters like this for certain attributes that are then used by the Leda Checker. This is because, in some cases, rules may be too strict. For example, when testing legacy code, some naming convention rules may be violated many times. You can deactivate these rules in the Checker or modify the rules online.

For example, the Reuse Methodology Manual (RMM) specifies that clock signals must have the prefix `clk_`. In the RMM policy, this rule is written as follows:

```
template CLOCK_ID is clock
    limit identifier to "^clk_"
end
```

This rule is often violated in code that was written before the RMM was used or that used another naming convention for clock signals. You can extend the rule definition using VRSL to include a variable parameter as follows:

```
template CLOCK_ID is clock
    limit identifier to "<clock_name>"
end
```

You then must specify a value for the <clock_name> macro for the Checker to read. You specify the value using a `rule_set_parameter` command in your `config.tcl` file. For more information, see the [Leda User Guide](#).

You can use predefined macros in regular expressions. For example, Leda replaces the predefined parameter <entity> in a regular expression with the name of the enclosing entity, if any. If you write the following rule:

```
--the architecture's name must be prefixed by the entity name
template AB is architecture_body
    limit identifier to "<entity>_[a-zA-Z0-9]+"
```

end

```
limit architecture_body to AB...
```

and have an entity called TOP, then Leda flags the architecture name RTL as an error, but accepts the name TOP_RTL. Similarly, you can use the following predefined parameters in regular expressions:

- architecture
- configuration

- package
- library
- component (name of component in component instantiation)
- formal (name of formal parameter in a map aspect)
- target (name of left-hand side in an assignment statement)

Constrain Max/Min Attributes to Predefined Values

You use max and min commands to control the number of elements in a given VHDL clause. For example, suppose you want ranges always to be descending downto 0 (for example, 5 downto 0...). You can write a rule to specify this as follows:

```
max low_bound in range is 0
```

You can also constrain a max/min attribute to a value defined in a macro. This rule then becomes a generic rule that can be used by multiple users, each with their own parameters. To change the value of the macro, use a `rule_set_parameter` command in your `config.tcl` file. For more information, see the [Leda User Guide](#).

The VRSL code itself stays the same, so you don't need to recompile the rule. For example, you could use a macro to implement the previous rule as follows:

```
max low_bound in range is "<range_low_value>"
```

and in the `config.tcl` file:

```
rule_set_parameter -rule rule_label -parameter range_low_value -value 0
```



Note

You must write the macros within inverted commas in the VRSL code, but do not treat them as regular expressions.

Linking to HTML Files

Rule developers often implement policies, or sets of rules, based on a formal document defining the coding standard (for example, the RMM). The VRSL language provides a way to specify a link to an HTML file with information derived from this formal document. This way, when a rule is violated, users can link directly from Leda Checker's Error Viewer to an HTML file with more information about each rule. This makes it easy for users to look up the reason for a particular rule, and perhaps find valid and invalid code examples and circuit diagrams that help explain the issue.

In other cases, the formal document describing a coding standard may not always be useful in the context of in-house conventions that vary somewhat from the external standards. To document such variations, the VRSL language provides a mechanism for linking to a second HTML document that is more like an application note, or current interpretation of a formal coding standard.

When you write rules using VRSL, you can code in the locations of one or both of these HTML documents, along with the error message to be printed when the rule is violated. You use two different environment variables as common prefixes to these file locations. The two environment variables are:

- Leda_HTML_DOC_PATH (for the standard interpretations)
- Leda_HTML_USR_PATH (for application notes that vary from standards)

If you want your rules to point to files stored locally, you can set these variables as follows:

```
% setenv Leda_HTML_DOC_PATH `file:/home/Standards/html/'
% setenv Leda_HTML_USR_PATH `file:/home/user/notes/html/'
```



Note

You must use the file: or http: prefix when you set these variables.

Leda supports the Netscape browser for viewing these help files.

The following VRSL example shows how to specify the location of HTML help documents:

```
limit file_name in architecture_body to "<entity>_<architecture>.vhd"
    message "Illegal file name for architecture"
        html_document "file.html#address"
        html_note     "note.html#address"
    severity error
```

Note that the HTML file locations are part of the message command. As such, they must only appear after the keyword “message” in a VRSL rule.

Parameterizing Error Messages

You can make error messages for Leda rules more informative by using parameters in the VRSL code for the message text. For example, suppose you have a rule that checks for redundant signals in sensitivity lists:

```
process (a1, a2, 14, 15, ..., a10)
  [ERROR] Redundant signal in sensitivity list
```

If you write this rule using a parameter that identifies the specific signal causing the error, sifting through the error messages and correcting the problems in your HDL source code becomes a simpler task. For example:

```
process (a1, a2, 14, 15, ..., a10)
  [ERROR] Signal a10 is not required in sensitivity list
```

The VRSL rule specification language supports a set of parameters that you can use in error messages to improve their readability and usefulness (see [Table 5](#)).

Table 5: Leda Error Message Parameters

Parameter	Function	Use With ...
<%item>	Returns the name of an attribute that is a valid RTL object (flip-flop, latch, clock, or reset) or HDL named object (declarative item, object, simple name, indexed name, slice name, or selected name). For example, use with any template that has an “identifier” attribute. Replace the identifier with the parameter in the error message text.	Any VRSL command (no, force, limit, set, max, min). For limitations on the use of <%item>, see “Limitations on use of <%item>” on page 33.
	Example VRSL code: no latch in all message “<%item> is inferred as a latch” severity error; For this rule, Leda reports the name of the latch output port in the error message.	
<%context>	Returns the name of the region/context in your HDL code where Leda flags an error. Note that if you specify the rule context as “all” in your VRSL code, Leda cannot determine a more specific context.	Any VRSL command (no, force, limit, set, max, min).
	Example VRSL code: no signal_decl in package_decl message “<%item> in <%context>: signals in packages not supported” severity error; For this rule, Leda reports the name of the VHDL package declaration in the error message.	

Table 5: Leda Error Message Parameters (Continued)

Parameter	Function	Use With ...
<%value>	Returns the value calculated for an item.	Set, max, or min commands.
	Example VRSL code: <pre>max states in fsm is 40 message" fsm with <%value> states: must be less than 40:" severity error;</pre> For this rule, Leda reports the actual number of states calculated for the FSM in the error message.	
<%formal> <%actual>	These parameters return the formal and actual names for items in association lists (for example, port maps and subprogram calls). Use with instantiations (components, modules) or subprogram calls.	Any VRSL command (no, force, limit, set, max, min).
	Example VRSL Code: <pre>limit port_connection in module_instantiation to T_G_521_11_1 message "Use same or similar names for ports (<%formal>) and signals (<%actual>)" severity warning;</pre> For this rule, Leda replaces <%formal> with the port name in the module definition and <%actual> with the associated signal.	

Limitations on use of <%item>

The <%item> parameter only works if the parameterized attribute is part of the command and not embedded in a template.

For example:

```
limit <attribute> in <context> to
```

and not:

```
template T is <context>
  limit <attribute> to ...
end
limit <context> to T.
```

Only the following attributes can be parameterized:

- alias_declaration
- asynchronous_reset
- attribute_declaration

- attribute_name
- clock
- complete_sensitivity
- component_declaration
- component_instantiation_statement
- configuration_declaration
- constant_declaration
- element_declaration
- entity_declaration
- file_declaration
- file_layout
- flipflop
- formal_parameter
- function_call
- gated_clock
- gated_reset
- generate_statement
- generic_declaration
- indexed_name
- interface_file_declaration
- interface_variable_declaration
- latch
- missing_signals_in_sensitivity_list
- mixed_async_sync_resetline
- multiplexed_clock
- object_definition
- operator
- package_declaration
- port_declaration

- `redundancy_in_sensitivity_list`
- `registered_outputs`
- `selected_name`
- `shared_variable_declaration`
- `signal_declaration`
- `simple_name`
- `slice_name`
- `subprogram_declaration`
- `subtype_declaration`
- `synchronous_reset`
- `type_declaration`
- `variable_declaration`

Formal Definition of VRSL

This section provides an abbreviated formal definition of VRSL syntax, using modified Backus Naur Form (BNF) notation to describe the VRSL grammar.

A VRSL description is a set of library units. There must be at least one ruleset unit, as this is where the commands are defined. This is summarized in the syntax:

```
policy ::= {leda_library_unit}
leda_library_unit ::= templateset | ruleset
```

Templateset

A templateset consists of a set of template declarations. It can also include other templatesets that contain commonly-used template declarations, similar to a VHDL package declaration. The grammar for a templateset is:

```
templateset ::= templateset unit_identifier is
                [template_include_part]
                [declarative_part]
            end templateset
template_include_part ::= use [templateset] unit_identifier
```

The identifier used in the `template_include_part` must be the name of a previously compiled templateset unit.

Ruleset

A ruleset consists of a set of template declarations and a set of commands. It can also include other templatesets, similar to a VHDL architecture. A ruleset is the only syntactical unit that you can use to define rules for a coding standard. The grammar for a ruleset is:

```
ruleset ::=ruleset unit_identifier is
                [template_include_part]
                [declarative_part]
                command_part
            end ruleset
```

Declarative Part

Nodes are internally defined for most VHDL_rules (rules based on the LRM). You can instantiate these nodes as templates in the declarative part of a templateset or a ruleset unit to add limitations to the VHDL_rule. The grammar for the declarative part of a ruleset is:

```

declarative_part ::= {template_declaration}
template_declaration ::= template template_identifier is template
                        {template_command_list}
                        end
template_command_list ::= template_decl_no_command
                        | template_decl_force_command
                        | template_decl_limit_command
                        | template_decl_set_command
                        | template_decl_max_or_min_command

```

Template identifiers are user-defined. You must specify unique template identifiers for each template.

Template No Command

The grammar for the template no command is:

```

template_decl_no_command ::= no full_attribute
full_attribute ::= template_kind
                 | local_attribute
                 | aggregate_attribute

```

The attribute kind must belong to the template identified in the template grammar rule in the immediately preceding template declaration.

Template Force Command

The grammar for the template force command is:

```

template_decl_force_command ::= force full_attribute

```

The attribute kind must belong to the template identified in the template grammar rule in the immediately preceding template declaration.

Template Limit Command

Template limit commands are divided into two types depending on whether the attribute to be limited is an aggregate attribute or not. The grammar for the template limit command is:

```

template_decl_limit_command ::= complete_template_decl_limit_command
                             aggregate_template_decl_limit_command
complete_template_decl_limit_command ::=
    limit template_kind to one_of_limit_list
aggregate_template_decl_limit_command ::=
    limit aggregate_attribute to template_aggregate_list
template_aggregate_list ::= first template_id_list
                          {then template_id_list }
                          [others template_id_list ]
one_of_limit_list ::= template_id_list
                   | string_list
id_list ::= identifier | identifier ',' id_list
string_list ::= "identifier" | "identifier" ',' string_list

```

You must first declare a template before listing it in the `template_id_list`. In addition, the attribute you specify for a template must belong to all nodes referenced in the `command_context`.

When an attribute is constrained to templates of type `identifier`, the string used in the `limit_id` attribute of this template can replace the `identifier` template itself. For example, the two commands below are identical.

```

template ID is identifier
    limit limit-id to "<entity>.vhd"
end

limit file_name in entity_declaration to ID...
limit file_name in entity_declaration to "<entity>.vhd"...

```

The `limit_list` can be a set of previously-defined template identifiers, a set of template kinds, or a set of strings. For example:

```

template SN is simple_name
end
template RE is binary_operation
    limit left_expression to SN
    limit right_expression to simple_name
end

```

In this VRSL template declaration example, the `left_expression` and `right_expression` attributes have the same limitations. They can only point to expressions of type `simple_name`. For `left_expression`, we declared a dummy template in the example that

matches all attributes of `simple_name` kind, whereas for `right_expression`, we used the template kind name directly. You cannot mix template identifiers and template kinds in the same list.

Template Set Command

The grammar for the template set command is:

```

template_decl_set_command ::= set set_attribute to set_value
set_value ::= STRING
           | number
           | enumerated_type_value
number ::= [-] DECIMAL_NUMBER
enumerated_type_value ::= evaluation_period
                       | literal_type
evaluation_period ::= unresolved
                  | locally_static_evaluation
                  | globally_static_evaluation
                  | dynamic_evaluation
literal_type ::= any_literal_type
              | abstract_literal_type
              | physical_literal_type
              | enumeration_literal_type
              | bit_string_literal_type

```

The `set_attribute` must belong to the template identified as `template_name` in the immediately preceding `template_declaration`. The `set_value` type must correspond to the type accepted by the attribute specified.

Template Max or Min Command

The grammar for the template max or min command is:

```

template_decl_max_or_min_command ::= template_decl_max_command
                                   | template_decl_min_command
template_decl_max_command ::= max max_min_attribute is number
template_decl_min_command ::= min max_min_attribute is number

```

The `max_min_attribute` must belong to the template identified as `template_name` in the immediately preceding `template_declaration`.

You use max and min commands to control the number of elements in a given VHDL clause. For example, suppose if you want a range always to be descending to 0 (for example, 5 downto 0...). You can write this rule as follows:

```
template DESCENDING_RANGE is range
  no      to
  max    low_bound is 0
  min    low_bound is 0
end
```

You can also use max and min commands to control the number of template kinds in a rule. For example; if you want only one wait statement per process, you could write:

```
template SEQ_PSS is process_statement
  max wait_statement is 1
end
```

Command Part

You define subset rules in the command part of a rule_file. The syntax for the command part is:

```
command_part ::= {command}
command ::= no_command
         | force_command
         | limit_command
         | set_command
         | max_or_min_command
```

No Command

The grammar for the no command is:

```
no_command ::= [label:] no no_command_context no_command_options
no_command_context ::= complete_no_command
                   | no_match_command
complete_no_command ::= full_attribute [command_context]
no_match_command ::= template_name in template_kind [of template_kind]
no_command_options ::= [message] [severity]
command_context ::= in context
context ::= all
           | template_kind
message ::= message "message_string" [html_doc [html_app]]
severity ::= severity severity_level
severity_level ::= note | warning | error | fatal
html_doc ::= "html_address_string"
html_app ::= "html_address_string"
```


You can use the `no` command to indicate that a defined template is not allowed in a given context. For example:

```
no wait_statement in subprogram_body
```

You can also use the `no` command to indicate that a fixed template is not allowed in a given context. For example:

```
template SUBP_WAIT is wait_statement
    no timeout
end
no SUBP_WAIT in wait_statement of subprogram_body
```

This example rule means that wait statements are allowed in subprogram bodies unless they match the template `SUBP_WAIT`. The attribute specified by `full_attribute` must belong to all template kinds referenced in the context clause.

If you do not specify the command context, the rule applies to all relevant template kinds by default.

Force Command

The grammar for the force command is:

```
force_command ::= [label :] force full_attribute command_options
command_options ::= [command_context ] [message] [severity]
```

The force command specifies that the corresponding clause must appear in the VHDL input code. The attribute specified by `full_attribute` must belong to all template kinds referenced in the context clause.

Limit Command

The grammar for the limit command is:

```

limit_command ::= complete_limit_command
                | aggregate_limit_command
complete_limit_command ::= [label :] limit template_kind
                        [command_context] to limit_list
                        [message] [severity]
                        {conditional_rule_block}
aggregate_limit_command ::= [label :] limit aggregate_attribute
                        [command_context] to template_aggregate_list
                        [message] [severity]
                        {conditional_rule_block}
conditional_rule_block ::= if identifier then {command} end if

limit_list ::= one_of_limit_list
            | allof allof_id_list
            | allof allof_template_list
one_of_limit_list ::= template_id_list
                 | string_list
all_of_id_list ::= identifier [message] | identifier [message] ','
all_of_id_list
all_of_template_list ::= template_kind [message] | template_kind
                    [message] ',' all_of_template_list

```

The attribute specified by `template_kind` must belong to all nodes referenced in the command context. You cannot define template identifiers and template kinds in the same list.

Set Command

The grammar for the set command is:

```

set_command ::= [label:] set set_attribute [command_context] to
              set_value [message] [severity]

```

When Leda finds the corresponding VHDL_rule in the input VHDL code, it matches the input code against the specified value. If they are not equal, Leda flags an error.

Max or Min Command

The grammar for the max or min command is:

```
max_or_min_command ::= max_command
                    | min_command
max_command ::= [label :] max max_min_attribute is number
              [message] [severity]
min_command ::= [label :] min max_min_attribute is number
              [message] [severity]
```

You use the max and min commands to control the number of elements in a given VHDL clause. For example, suppose you want a range always to be descending to 0 (for example, 5 downto 0...). You can write this rule as follows:

```
max low_bound in range is 0
```

You can also use max and min commands to limit the number of template kinds in a rule. For example; if you wanted only one wait statement per process, you could write the following rule:

```
max wait_statement in process_statement is 1
```

VRSL Primitives

The following primitive types are defined in VRSL:

- [“Literal_Type” on page 44](#)
- [“Evaluation_Periods” on page 45](#)
- [“Error_Status” on page 45](#)
- [“Recursion_Type” on page 45](#)
- [“Edge_Type” on page 46](#)

Literal_Type

Enumerates the different types that a literal value can take. This enumerated type is pointed to by the `value_type` attribute of the literal template. The `literal_type` values are:

- `Any_Literal_Type`—An abstract literal can only be a decimal or based number. For example: 12, 123-456, 12-0, 6.023E+24, 2#11, 16#F.FF#E+2
- `Abstract_Literal_Type`—A physical literal is an abstract literal followed by a unit name. For example: 10ns.
- `Physical_Literal_Type`—A physical literal is an abstract literal followed by a unit name; for example: 10ns.
- `Enumerated_Literal_Type`—An enumerated literal is an element of an enumeration type. For example: TRUE or '1'.
- `String_Literal_Type`—A string literal is a string of characters. For example: “hello world”.
- `Bit_String_Literal_Type`—A bit string literal is a group of character literals. For example: “1111” or B “0011”.

Evaluation_Periods

Enumerates the different evaluation times for expressions. This enumerated type is pointed to by the `evaluation_time` attribute. The `evaluation_periods` values are:

- Undefined
- `Locally_Static_Evaluation`—Locally static expressions are those that can be evaluated at compile time. For example: `1+2`.
- `Globally_Static_Evaluation`—Globally static expressions are those that can only be evaluated after elaboration when generic values are known and functions have been executed. For example: `foo(3)`.
- `Dynamic_Evaluation`—Dynamic expressions are those that can only be evaluated during simulation. For example, those that include signals or variables.

Error_Status

Enumerates the different severity levels that a command can use. The `error_status` values are:

- Note
- Warning
- Error
- Fatal

Recursion_Type

Enumerates the different recursion types allowed by the attribute `recursion_type`. The `recursion_type` values are:

- `all_recursion`
- `static_recursion`
- `no_recursion`

Edge_Type

Enumerates the different values the edge attribute can have. This attribute is used by clock and reset nodes to model the active edge of the corresponding signal. The edge_type values are:

- Undefined_Edge
- Rising_Edge
- Falling_Edge
- Both_Edges
- High_Level
- Low_Level

VRSL Classes

Each VRSL template belongs to one or more classes, or different types of VHDL code. Classes are also used to specify a group of template kinds that are valid for a given attribute. Each template belongs to one or more classes. Classes are used to indicate a group of template kinds that are valid for a given attribute.

There is one important exception. The class ID only includes one template, the identifier template. However, attributes of type ID may also accept string literals as values. For example, for the following template description:

```
template STD_BIT_ID is identifier
    limit limit_id to "STD.STANDARD.BIT"
end
template STD_BIT is type_declaration
    limit identifier to STD_BIT_ID
end
```

the attribute identifier is of class ID, so you also could have written template STD_BIT as:

```
template STD_BIT is type_declaration
    limit identifier to "STD.STANDARD.BIT"
end
```

The following classes are defined in VRSL:

- [“CONCURRENT_STATEMENT” on page 48](#)
- [“DECLARATIVE_ITEM” on page 48](#)
- [“DISCRETE_RANGE” on page 49](#)
- [“DRIVEN_OBJECT” on page 49](#)
- [“EXPRESSION” on page 49](#)
- [“ID” on page 50](#)
- [“NAME” on page 50](#)
- [“NAMED_ENTITY” on page 50](#)
- [“OBJECT_ITEM” on page 51](#)
- [“REGION_PART” on page 51](#)
- [“SEQUENTIAL_STATEMENT” on page 52](#)
- [“STATEMENT” on page 53](#)
- [“TARGET” on page 54](#)

- [“TYPE_MARK” on page 54](#)
- [“TYPE_DEFINITION” on page 54](#)

CONCURRENT_STATEMENT

The CONCURRENT_STATEMENT class includes the following templates:

- concurrent_assertion_statement
- block_statement
- component_instantiation_statement
- conditional_signal_assignment
- generate_statement
- procedure_call_statement
- process_statement
- selected_signal_assignment

DECLARATIVE_ITEM

The DECLARATIVE_ITEM class includes the following templates:

- alias_declaration
- attribute_declaration
- attribute_specification
- component_declaration
- constant_declaration
- disconnection_specification
- file_declaration
- group_declaration
- group_template_declaration
- name_prefix
- shared_variable_declaration
- signal_declaration
- subprogram_body

- subprogram_declaration
- subtype_declaration
- type_declaration
- use_clause
- variable_declaration

DISCRETE_RANGE

The DISCRETE_RANGE class includes the following templates:

- subtype_indication
- range

DRIVEN_OBJECT

The DRIVEN_OBJECT class includes the following templates:

- port_declaration
- formal_parameter
- signal_declaration

EXPRESSION

The EXPRESSION class includes the following templates:

- aggregate
- allocator
- binary_operation
- expression
- function_call
- literal

SUBCLASS: NAME

The NAME subclass includes the following templates:

- _m attribute_name
- _m indexed_name

- m selected_name
- m simple_name
- m slice_name

ID

The ID class includes the following template:

- identifier

NAME

The NAME class includes the following templates:

- attribute_name
- indexed_name
- selected_name
- simple_name
- slice_name

NAMED_ENTITY

The NAMED_ENTITY class includes the following templates:

- design_unit
- function_call

SUBCLASS: TYPE_MARK

The TYPE_MARK subclass includes all VRSL templates.

SUBCLASS: OBJECT_ITEM

The OBJECT_ITEM subclass includes all VRSL templates.

SUBCLASS: DECLARATION_ITEM

The DECLARATION_ITEM subclass includes all VRSL templates.

OBJECT_ITEM

The OBJECT_ITEM class includes the following templates:

- alias_declaration
- architecture_body
- attribute_declaration
- component_declaration
- constant_declaration
- configuration_declaration
- entity_declaration
- file_declaration
- formal_parameter
- interface_constant_declaration
- interface_file_declaration
- name_prefix
- port_declaration
- generic_declaration
- package_declaration
- shared_variable_declaration
- signal_declaration
- subprogram_declaration
- variable_declaration

REGION_PART

The REGION_PART class includes the following templates:

- architecture_body
- block_configuration
- block_specification
- block_statement
- component_declaration

- configuration_declaration
- entity_declaration
- for_loop_statement
- generate_statement
- loop_statement
- package_body
- package_declaration
- process_statement
- subprogram_body
- subprogram_declaration
- while_loop_statement

SEQUENTIAL_STATEMENT

The SEQUENTIAL_STATEMENT class includes the following templates:

- assertion_statement
- case_statement
- exit_statement
- for_loop_statement
- if_statement
- loop_statement
- next_statement
- procedure_call_statement
- report_statement
- return_statement
- signal_assignment_statement
- variable_assignment_statement
- wait_statement
- while_loop_statement

STATEMENT

The STATEMENT class includes the following subclasses:

- CONCURRENT_STATEMENT

The CONCURRENT_STATEMENT subclass includes the following templates

- m concurrent_assertion_statement
- m block_statement
- m component_instantiation_statement
- m conditional_signal_assignment
- m generate_statement
- m procedure_call_statement
- m process_statement
- m selected_signal_assignment

- SEQUENTIAL_STATEMENT

The SEQUENTIAL_STATEMENT subclass includes the following templates

- m assertion_statement
- m case_statement
- m for_loop_statement
- m if_statement
- m loop_statement
- m procedure_call_statement
- m report_statement
- m signal_assignment_statement
- m variable_assignment_statement
- m wait_statement
- m while_loop_statement

TARGET

The TARGET class includes the following templates:

- aggregate
- indexed_name
- selected_name
- simple_name
- slice_name

TYPE_MARK

The TYPE_MARK class includes the following templates:

- identifier
- name_prefix
- type_declaration
- subtype_declaration

Note that attributes of type TYPE_MARK can also take string literals as parameters.

TYPE_DEFINITION

The TYPE_DEFINITION class includes the following templates:

- access_type_definition
- constrained_array_definition
- enumeration_type_definition
- file_type_definition
- floating_type_definition
- integer_type_definition
- physical_type_definition
- record_type_definition
- unconstrained_array_definition

2

Templates and Attributes

Introduction

This chapter presents an API-like reference for the VRSL templates and attributes, organized in the following major sections:

- [“Chip-Level Templates and Attributes” on page 55](#)
- [“Block-Level Templates and Attributes” on page 66](#)

Chip-Level Templates and Attributes

Leda applies chip-level rules to the entire design hierarchy, whereas it applies block-level rules to each unit or module individually. You can write chip-level rules using the templates and attributes described in this section. Note that there are also some block-level templates, such as `clock` and `synchronous_initialization`, which contain attributes that you can also use for writing chip-level rules. Reference information for the chip-level templates and attributes is presented in the following subsections, one for each template:

- [“Connectivity Path Template” on page 56](#)
- [“Data Signal Template” on page 58](#)
- [“Design Template” on page 58](#)
- [“Flipflop Template” on page 60](#)
- [“Latch Template” on page 62](#)
- [“Logic Cost Template” on page 63](#)
- [“Test Signal Template” on page 65](#)

Connectivity Path Template

A lot of rules concern what appears on the path of control signals. This can be anything from gated clocks, to counting the number of inverters, to forbidding buffers. Each control signal therefore has a `connectivity_path` attribute which points to a set of templates of this type. Some design for test (DFT) rules concern what appears on the path of data or control signals. Also, some rules imply the analysis of a data signal path; thus, a `connectivity_path` attribute is added to each data signal. The `connectivity_path` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 6](#).

Table 6: connectivity_path Template

Attribute	Kind	Limit_Kind
<code>buffer_count</code>	max/min	N/A
<code>inverter_count</code>	max/min	N/A
<code>is_combinatorial</code>	local	N/A
<code>starting_unit</code>	template	REGION_PART
<code>data</code>	local	N/A
<code>flipflop_as_source</code>	local	N/A
<code>latch_as_source</code>	local	N/A
<code>is_reset</code>	local	N/A
<code>control_src_count</code>	max/min	N/A
<code>within_same_clkdomain</code>	local	N/A

- Use the `starting_unit` attribute to constrain if there is any control or data on.
- Use the `data` attribute to constrain if there is any data on the connectivity path.
- Use the `flipflop_as_source` attribute to constrain a signal to be driven by a flip-flop output signal. This attribute was used to specify the DFT rules TEST_980 and TEST_981.
- Use the `latch_as_source` attribute to constrain a signal to be driven by a latch output signal. This attribute was used to specify the DFT rules TEST_974, TEST_975, TEST_978, and TEST_979.
- Use the `is_reset` attribute to constrain the clock signal to be also a reset. This attribute was used to specify the DFT rule TEST_994.

- Use the `control_src_count` attribute to constrain the number of clock signals that control a register. This attribute was used to specify the DFT rules `TEST_976` and `TEST_977`.
- Use the `within_same_clkdomain` attribute to constrain the path between two registers to be within the same clock domain. Combined with the `flipflop_as_source` or `latch_as_source` attributes, this attribute was used to specify the DFT rules `TEST_974`, `TEST_975`, and `TEST_978` through `TEST_981`.

Connectivity Path Example

Here is an example rule written in VRSL that uses the `connectivity_path` template:

```
TEST_974 : avoid latch enabled by clock clk which affects data input of
latch on the same clock.

template CN974 is connectivity_path
    force within_same_clkdomain
    no latch_as_source
end

template D974 is data_signal
    limit connectivity_path to CN974
end

template L974 is latch
    limit data_signal to D974
end

TEST_974:
limit latch in design to L974
    message "Latch as source and latch as destination of data path on the
    same clock is not allowed"
severity ERROR
```

Data Signal Template

Just as the clock, asynchronous_initialization, and synchronous_initialization templates constrain clocks and resets for registers, the data_signal template constrains the data path to registers. Each register (flip-flop, latch) therefore has a data_signal attribute that you can use to write design for test (DFT) rules. The data_signal template has an attribute named connectivity_path just like the clock, asynchronous_initialization, and synchronous_initialization templates. The data_signal template is a secondary template belonging to no classes. It contains the attributes shown in [Table 7](#).

Table 7: data_signal Template

Attribute	Kind	Limit_Kind
connectivity_path	template	connectivity_path
fixed_value	local	N/A

Use the fixed_value attribute to detect data with inputs that have fixed values.

Design Template

The design template is the basic chip-level template. You can only use the local attributes in this template with the force and no commands. The design template also contains attributes that you can use to limit clocks and resets, and build other more complex rules. Insert the design template at the top level of a ruleset just like all primary templates (template ruleset has attribute design which points to templates of this type). The design template is a primary template belonging to no classes. It contains the attributes shown in [Table 8](#).

Table 8: design Template

Attribute	Kind	Limit_Kind
asynchronous_initialization	template	asynchronous_initialization
clock	template	clock
synchronous_initialization	template	synchronous_initialization
top_unit	template	REGION_PART
asynchronous_feedback	local	N/A
asynchronous_logic	local	N/A
drivers_per_signal	max/min	N/A

Table 8: design Template (Continued)

Attribute	Kind	Limit_Kind
non_tristate_drivers_per_signal	max/min	N/A
pulse_generator	local	N/A
meta_stability	local	N/A
flipflop	template	flipflop
gated_clock	local	N/A
clock_count	max/min	N/A
latch	template	latch
mixed_clock	local	N/A
load_count	max/min	N/A
multiplexed_clock	local	N/A
reset_count	max/min	N/A
gated_reset	local	N/A
initialization_count	max/min	N/A
mixed_async_sync_line	max/min	N/A
glue_logic_at_top	local	N/A
registered_outputs	local	N/A
registered_inputs	local	N/A
set_count	max/min	N/A
sync_ff_count	max_min	N/A
comb_cost	limit	logic_cost
logic_level	limit	logic_cost

- Use the top_unit attribute to constrain the top-level unit (for example, control its name or its containing file).
- Use the meta_stability attribute to make sure that there are at least two consecutive flip-flops on the data flow path when changing clock domains.

- Use the `gated_clock` attribute to constrain the presence of gated clocks in the whole design. If you want to allow gated clocks in specific units, use clock templates instead.
- Use the `comb_cost` and `logic_level` attributes to associate an identical cost to each operation in the logic cost template and make sure the `max_cost` does not exceed the specified threshold.
- Use the `flip-flop` and `latch` attributes to constrain the flip-flops and latches in the design. This can be a constraint of its `connectivity_path` attribute to specify a DFT rule.
- Use the `mixed_async_sync_resetline` attribute to detect reset lines that are used as both synchronous and asynchronous register resets.
- Use the `sync_ff_count` attribute to constrain the flip-flop synchronizer number for metastability rule checks. The default flip-flop synchronizer number is two. If you want a different number of synchronizers, write the rule as shown in the following example:

```

template META is design
    max sync_ff_count is 4
    min sync_ff_count is 4
end
limit design to META
message "needs 4 synchronizers"
severity error

```

Flipflop Template

The flipflop template is a primary template belonging to no classes. It contains the attributes shown in [Table 9](#).

Table 9: flipflop Template

Attribute	Kind	Limit_Kind
<code>asynchronous_initialization</code>	template	<code>asynchronous_initialization</code>
<code>cased_identifier</code>	template	ID
<code>identifier</code>	template	ID
<code>tinput</code>	template	EXPRESSION
<code>output</code>	template	EXPRESSION
<code>object_definition</code>	template	OBJECT_ITEM
<code>synchronous_initialization</code>	template	<code>synchronous_initialization</code>

Table 9: flipflop Template (Continued)

Attribute	Kind	Limit_Kind
clock	template	clock
data_signal	template	data_signal
has_clock_as_data	local	N/A

- Use the `has_clock_as_data` attribute to constrain a clock signal to be a clock and data input signal of the same flip-flop. For example, DFT rule `TEST_972` differs from `TEST_970`, where a clock is a data input of another flip-flop. `TEST_970` is specified using the `data` attribute of the `connectivity_path` attached to the clock template.

Latch Template

The latch template is a primary template belonging to no classes. It contains the attributes shown in [Table 10](#).

Table 10: latch Template

Attribute	Kind	Limit_Kind
asynchronous_initialization	template	asynchronous_initialization
cased_identifier	template	ID
identifier	template	ID
input	template	EXPRESSION
output	template	EXPRESSION
object_definition	template	OBJECT_ITEM
synchronous_initialization	template	synchronous_initialization
clock	template	clock
data_signal	template	data_signal
has_clock_as_data	local	N/A

Use the `has_clock_as_data` attribute to constrain a clock signal to be a clock and data input signal of the same latch. For example, DFT rule `TEST_973` differs from `TEST_971`, where a clock is a data input of another latch. `TEST_971` is specified using the `data` attribute of the `connectivity_path` attached to the clock template.

Logic Cost Template

The `logic_cost` template does not correspond to any part of the formal definition of the Verilog language. Instead, you use this template to constrain the logic in hardware logic blocks. The `logic_cost` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 11](#).

Table 11: `logic_cost` Template

Attribute	Kind	Limit_Kind
<code>max_cost</code>	set	<integer>
<code>multiply_cost</code>	set	<integer>
<code>divide_cost</code>	set	<integer>
<code>modulus_cost</code>	set	<integer>
<code>abs_cost</code>	set	<integer>
<code>plus_cost</code>	set	<integer>
<code>minus_cost</code>	set	<integer>
<code>remainder_cost</code>	set	<integer>
<code>power_cost</code>	set	<integer>
<code>and_cost</code>	set	<integer>
<code>nand_cost</code>	set	<integer>
<code>or_cost</code>	set	<integer>
<code>nor_cost</code>	set	<integer>
<code>xor_cost</code>	set	<integer>
<code>xnor_cost</code>	set	<integer>
<code>comparator_cost</code>	set	<integer>
<code>mux_cost</code>	set	<integer>
<code>buffer_cost</code>	set	<integer>
<code>function_cost</code>	set	<integer>
<code>shift_cost</code>	set	<integer>
<code>decoder_cost</code>	set	<integer>
<code>reset_at_hierarchical_boundary</code>	local	N/A

Use the cost attributes with the set command to constrain hardware logic blocks to specified levels for the various items. For example, you can use this template to build rules that constrain the number of arithmetic operators used in a logic block. Note that cost in the context of the logic_cost template is a generic term that can refer to code complexity, timing, area, or power, depending on the attribute used and your application of the rule.

For example, if you want to write a rule that prohibits the use of complex arithmetic if it includes multiplication expressions similar to the following:

```
a = b*c+d
```

you can set the multiply_cost operator to be extremely high without regard to the other operators as follows:

```
template MY_COSTS is logic_cost
  set max_cost to 100
  set multiply_cost to 100
  set adder_cost to 0
end
MYRULE_1:
limit comb_logic in design to MY_COSTS
...
```

This rule fires if it finds a multiplication operator (*) in the hardware logic block.

Use the max_cost attribute to set a total cost for the sum of all other cost attributes specified in a rule.

Test Signal Template

A lot of design for test (DFT) rules concern what appears on the path of test control signals, especially the rules detecting errors which prevent scan insertion (labelled TEST_953, TEST_954, and TEST_963 through TEST_969). The asynchronous_initialization, synchronous_initialization, and clock templates therefore all have a test_signal attribute which you can use to constrain the test clock and test reset signals. This allows you to specify rules that check the controllability of a register (flip-flop or latch) for DFT.

The test_signal template is a secondary template belonging to no classes. It contains the attributes shown in [Table 12](#).

Table 12: test_signal Template

Attribute	Kind	Limit_Kind
control_at_start	local	N/A
disable_control	local	N/A
hold_latch_data	local	N/A
reach_memory	local	N/A

- Use the control_at_start attribute to constrain a test clock signal to control a register at the beginning of the cycle. This attribute was used to write the DFT rules TEST_963 and TEST_964.
- Use the disable_control attribute to constrain a test reset signal to be able to disable the register reset control. This attribute was used to specify the DFT rules TEST_968 and TEST_969.
- Use the hold_latch_data attribute to ensure that the test clock signal holds data in latches at the beginning of the cycle. This attribute was used to program the DFT rule TEST_965.
- Use the reach_memory attribute to ensure that the test clock signal and the test reset signal are able to reach control signals of registers.

Test Signal Template Example

Here is an example rule written in VRSL that uses the test_signal template:

```
TEST_953: Flipflop's clock is not reached by any test clock
template TS953 is test_signal
    force reach_memory
end

template CK953 is clock
    limit test_signal to TS953
end

template FF953 is flipflop
    limit clock to CK953
end

TEST_953:
limit flipflop in design to FF953
    message "Flipflop is not reached by any test clock"
        severity ERROR
```

Block-Level Templates and Attributes

You use the templates and attributes described in this section to develop coding rules that operate on the HDL block or module level, in contrast to the chip-level templates and attributes, which you use to develop rules that work on the entire design hierarchy. There are two main types of block-level rules that you can build using these templates and attributes:

- **Language-based rules**—use to constrain different VHDL constructs to ensure that they correspond to acceptable values, ranges, or conventions.
- **Hardware-based rules**—use to control the hardware semantics of VHDL. Certain VHDL code results in specific hardware features when you synthesize the descriptions (for example, latches, flip-flops, and finite state machines). You can build hardware-based rules to check that inferred hardware in your design is used correctly.

Reference information for the block-level templates and attributes is presented in the following subsections, one for each template:

- [“Access Type Definition Template” on page 70](#)
- [“Aggregate Template” on page 70](#)
- [“Alias Declaration Template” on page 71](#)
- [“Allocator Template” on page 72](#)

- “Architecture Body Template” on page 73
- “Assertion Statement Template” on page 77
- “Association Element Template” on page 78
- “Association List Template” on page 79
- “Asynchronous Initialization Template” on page 80
- “Attribute Declaration Template” on page 81
- “Attribute Name Template” on page 82
- “Attribute Specification Template” on page 83
- “Binary Operation Template” on page 85
- “Binding Indication Template” on page 86
- “Block Configuration Template” on page 87
- “Block Specification Template” on page 88
- “Block Statement Template” on page 89
- “Case Statement Template” on page 91
- “Clock Template” on page 93
- “Comment Template” on page 94
- “Component Configuration Template” on page 95
- “Component Declaration Template” on page 95
- “Component Instantiation Statement Template” on page 97
- “Component Specification Template” on page 98
- “Concurrent Assertion Statement Template” on page 99
- “Concurrent Procedure Call Statement Template” on page 100
- “Conditional Signal Assignment Template” on page 101
- “Conditional Waveforms Template” on page 102
- “Configuration Declaration Template” on page 104
- “Configuration Specification Template” on page 106
- “Constant Declaration Template” on page 106
- “Constrained Array Definition Template” on page 107
- “Design Unit Template” on page 108

- “Disconnection Specification Template” on page 109
- “Element Association Template” on page 109
- “Element Declaration Template” on page 110
- “Entity Declaration Template” on page 111
- “Enumeration Type Definition Template” on page 115
- “Exit Statement Template” on page 116
- “Expression Template” on page 117
- “FSM Template” on page 118
- “File Declaration Template” on page 119
- “File Layout Template” on page 120
- “File Type Definition Template” on page 121
- “Flipflop Template” on page 121
- “Floating Type Definition Template” on page 122
- “For Loop Statement Template” on page 123
- “Formal Parameter Template” on page 125
- “Function Call Template” on page 126
- “Generate Statement Template” on page 128
- “Generic Declaration Template” on page 130
- “Group Declarations” on page 131
- “Group Template Declarations” on page 131
- “Header Comment Template” on page 132
- “Identifier Template” on page 133
- “If Statement Template” on page 134
- “Indexed Name Template” on page 136
- “Integer Type Definition Template” on page 137
- “Interface File Declaration Template” on page 138
- “Interface Variable Declaration Template” on page 139
- “Latch Template” on page 140
- “Literal Template” on page 141

- “Loop Statement Template” on page 143
- “Next Statement Template” on page 145
- “Package Body Template” on page 146
- “Package Declaration Template” on page 148
- “Physical Type Definition Template” on page 150
- “Port Declaration Template” on page 151
- “Procedure Call Statement Template” on page 154
- “Process Statement Template” on page 155
- “Range Template” on page 159
- “Record Type Definition Template” on page 161
- “Report Statement Template” on page 161
- “Return Statement Template” on page 162
- “Selected Name Template” on page 163
- “Selected Signal Assignment Template” on page 164
- “Selected Waveforms Template” on page 165
- “Shared Variable Declaration Template” on page 166
- “Signal Assignment Statement Template” on page 167
- “Signal Declaration Template” on page 168
- “Simple Name Template” on page 170
- “Slice Name Template” on page 171
- “Statement Format Template” on page 172
- “Subprogram Body Template” on page 173
- “Subprogram Declaration Template” on page 178
- “Subtype Declaration Template” on page 180
- “Subtype Indication Template” on page 181
- “Synchronous Initialization Template” on page 182
- “Type Declaration Template” on page 183
- “Unconstrained Array Definition Template” on page 184
- “Use Clause Template” on page 185

- “Variable Assignment Statement Template” on page 185
- “Variable Declaration Template” on page 186
- “Wait Statement Template” on page 187
- “Waveform Element Template” on page 188
- “Waveform Template” on page 189
- “While Loop Statement Template” on page 189

Access Type Definition Template

The LRM (§3.3.2) defines the grammar for access type definition as follows:

```
access_type_definition ::= access subtype_indication
```

Access types are associated with the allocation and deallocation of objects. The `access_type_definition` template is a primary template belonging to no classes. It contains the attribute shown in [Table 13](#).

Table 13: access_type_definition Template

Attribute	Kind	Limit_Kind
subtype_indication	template	subtype_indication

Aggregate Template

The LRM (§7.3.2) defines the grammar for aggregate as follows:

```
aggregate ::= ( element_association { , element_association } )
```

The aggregate template is a primary template belonging to the `EXPRESSION` class. It contains the attributes shown in [Table 14](#).

Table 14: aggregate Template

Attribute	Kind	Limit_Kind
choice	template	EXPRESSION
positional_association	local	N/A
named_association	local	N/A
multiple_choices	local	N/A
record_aggregate	local	N/A

Table 14: aggregate Template (Continued)

Attribute	Kind	Limit_Kind
others	local	N/A
element_count	max/min	N/A
element_association_s	aggregate	element_association

You can use the choice attribute and any of the local attributes to constrain every element_association template. Or, you can add these constraints to each element_association template individually to achieve the same effect.

Alias Declaration Template

The LRM (§4.3.3) defines the grammar for alias declaration as follows:

```
alias_declaration ::=
  alias alias_designator [ : subtype_indicator ] is name [ signature ] ;
  alias_designator ::= identifier | character_literal | operator_symbol
```

The alias_declaration template is a primary template belonging to the OBJECT_ITEM class. It contains the attributes shown in [Table 15](#).

Table 15: alias_declaration Template

Attribute	Kind	Limit_Kind
identifier	template	ID
subtype_indication	template	subtype_indication
name	template	NAME
declarative_region	template	REGION_PART
cased_identifier	template	ID
character_literal	local	N/A
operator_symbol	local	N/A
one_declaration_per_line	local	N/A
comment	local	N/A

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is alias_declaration
    limit identifier to "[a-z][a-z0-9_]*$"
end
template T2 is alias_declaration
    limit cased_identifier to "[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

Allocator Template

The LRM (§7.3.6) defines the grammar for allocator as follows:

```
allocator ::= new subtype_indication | new qualified_expression
```

The allocator template is a secondary template belonging to the `EXPRESSION` class. It contains the attributes shown in [Table 16](#).

Table 16: allocator Template

Attribute	Kind	Limit_Kind
<code>subtype_indication</code>	template	<code>subtype_indication</code>
<code>expression</code>	template	<code>EXPRESSION</code>

Architecture Body Template

The LRM (§1.2) defines the grammar for architecture body as follows:

```
architecture_body ::=
    architecture identifier of entity_name is
        architecture_declarative_part
    begin
        architecture_statement_part
    end [ architecture ] [ architecture_simple_name ] ;
```

The LRM (§1.2.1) defines the grammar for the architecture declarative part as follows:

```
architecture_declarative_part ::=
    { block_declarative_item }
block_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | configuration_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration
```

The LRM (§1.2.2) defines the grammar for the architecture statement part as follows:

```
architecture_statement_part ::= { concurrent_statement }
```

Architecture bodies define the body of a design entity. The `architecture_body` template is a primary template belonging to no classes. It contains the attributes shown in [Table 17](#).

Table 17: architecture_body Template

Attribute	Kind	Limit_Kind
identifier	template	ID
cased_identifier	template	ID
subprogram_declaration	template	subprogram_declaration
subprogram_body	template	subprogram_body

Table 17: architecture_body Template (Continued)

Attribute	Kind	Limit_Kind
type_declaration	template	type_declaration
statement_format	template	statement_format
subtype_declaration	template	subtype_declaration
constant_declaration	template	constant_declaration
related_unit	local	N/A
shared_variable_declaration	template	shared_variable_declaration
signal_declaration	template	signal_declaration
file_declaration	template	file_declaration
alias_declaration	template	alias_declaration
component_declaration	template	component_declaration
attribute_declaration	template	attribute_declaration
attribute_specification	template	attribute_specification
configuration_specification	template	configuration_specification
disconnection_specification	template	disconnection_specification
use_clause	template	use_clause
group_template_declaration	template	group_template_declaration
group_declaration	template	group_declaration
concurrent_assertion_statement	template	concurrent_assertion_statement
concurrent_procedure_call_statement	template	concurrent_procedure_call_statement
process_statement	template	process_statement
block_statement	template	block_statement
selected_signal_assignment	template	selected_signal_assignment
conditional_signal_assignment	template	conditional_signal_assignment
component_instantiation_statement	template	component_instantiation_statement
generate_statement	template	generate_statement

Table 17: architecture_body Template (Continued)

Attribute	Kind	Limit_Kind
design_unit	template	design_unit
header_comment	template	header_comment
entity_declaration	template	entity_declaration
top_architecture	template	architecture_body
file_name	template	ID
ncs_file_name	template	ID
file_layout	template	file_layout
unused_declaration	local	N/A
line_count	max/min	N/A
file_length	max/min	N/A
clock_signal	max/min	N/A
synchronous_reset_signal	max/min	N/A
asynchronous_reset_signal	max/min	N/A
statement_profile_s	aggregate	CONCURRENT_STATEMENT
declaration_profile_s	aggregate	DECLARATIVE_ITEM
fsm	template	fsm

Employ the `use_clause` attribute to limit `use_clauses` inside the declaration. To constrain `use clauses` (and `library clauses`) appearing before the declaration, use the `design_unit` template instead.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the `entity_declaration` attribute to constrain the architecture entity.
- Use the `top_architecture` attribute to constrain the top-level architecture. To make this work, you must specify the name of the top-level architecture using the `-top` option of the command-line Checker.

- Use the `file_name` attribute to configure the name of the containing file. For example:

```
limit file_name in architecture_body to "<architecture>.vhd"
```

- Use the `unused_declaration` attribute to test for unused declarations. For example:
- ```
no unused_declaration in architecture_body
```
- Use the `line_count` attribute to specify the maximum or minimum lines of code in a unit.
  - Use the `clock_signal` attribute to specify the maximum or minimum number of clock signals in an architecture.
  - Use the `asynchronous_reset_signal` attribute to specify the maximum or minimum number of asynchronous reset signals in an architecture.
  - Use the `synchronous_reset_signal` attribute to specify the maximum or minimum number of synchronous reset signals in an architecture.
  - Use the `statement_profile_s` attribute to specify the order of statements. For example:

```
limit statement_profile_s in architecture_body to
 first COMP_INST_STM
 then PROCESS_STM
```

- Use the `declaration_profile_s` attribute to specify the order of declarations. For example:

```
limit declaration_profile_s in architecture_body to
 first type_declaration
 then subtype_declaration
 then {}
```

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases when the identifier is case-sensitive. Consider the following templates:

```
template T1 is architecture_body
 limit identifier to "^ [a-z] [a-z0-9_]*$"
end
template T2 is architecture_body
 limit cased_identifier to "^ [a-z] [a-z0-9_]*$"
end
```

The identifier `id1` matches both templates, whereas the identifier `ID1` only matches the first template. The attribute `identifier` is not case-sensitive because VHDL is not case-sensitive.

- Use the `related_unit` attribute to check that the architecture or the configuration you can find in a file is related to the entity described in that same file.

## Assertion Statement Template

The LRM (§8.2) defines the grammar for assertion statement as follows:

```
assert_statement ::= [label :] assertion ;
assertion ::= assert condition
 [report expression]
 [severity expression]
```

The `assertion_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 18](#).

**Table 18: `assertion_statement` Template**

| Attribute           | Kind     | Limit_Kind  |
|---------------------|----------|-------------|
| label               | template | ID          |
| cased_label         | template | ID          |
| condition           | template | EXPRESSION  |
| report_expression   | template | EXPRESSION  |
| severity_expression | template | EXPRESSION  |
| declarative_region  | template | REGION_PART |
| postponed           | local    | N/A         |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is report_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is report_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Association Element Template

The LRM (§4.3.2.2) defines the grammar for association element as follows:

```

association_element ::= [formal_part =>] actual_part
formal_part ::=
 formal_designator
 | function_name (formal_designator)
 | type_mark (formal_designator)
formal_designator ::= generic_name | port_name | parameter_name
actual_part ::=
 actual_designator
 | function_name (actual_designator)
 | type_mark (actual_designator)
actual_designator ::=
 expression
 | signal_name
 | variable_name
 | open

```

The `association_element` template is a primary template belonging to no classes. It contains the attributes shown in [Table 19](#).

**Table 19: association\_element Template**

| Attribute              | Kind     | Limit_Kind |
|------------------------|----------|------------|
| actual_designator      | template | EXPRESSION |
| named_association      | local    | N/A        |
| positional_association | local    | N/A        |
| range_overflow         | local    | N/A        |
| open                   | local    | N/A        |
| others                 | local    | N/A        |
| formal_function        | local    | N/A        |
| formal_type_mark       | local    | N/A        |
| actual_function        | local    | N/A        |
| actual_type_mark       | local    | N/A        |

Some of the attributes in association list templates also appear in association element templates, allowing finer control over these elements. To force positional (or named) association, you can use a command like the following:

```
force positional_association in association_element
```

Use the `formal_function` and `formal_type_mark` attributes to force or prohibit the presence of function calls or type conversions in formal parameters.

Use the `actual_function` and `actual_type_mark` attributes to force or prohibit the presence of function calls or type conversions in actual parameters.

Use the `open` attribute to force or prohibit the use of the “open” keyword in actual parameters.

## Association List Template

The LRM (§4.3.2.2) defines the grammar for association list as follows:

```
association_list ::= association_element { , association_element }
```

When you call a subprogram or instantiate a component, you use VHDL association lists to establish the correspondence between formal or local generic, port, or parameter names on the one hand, and local or actual names or expressions on the other.

The `association_list` template is a primary template belonging to the no classes. It contains the attributes shown in [Table 19](#).

**Table 20: association\_list Template**

| Attribute                           | Kind      | Limit_Kind                       |
|-------------------------------------|-----------|----------------------------------|
| <code>named_association</code>      | local     | N/A                              |
| <code>positional_association</code> | local     | N/A                              |
| <code>open</code>                   | local     | N/A                              |
| <code>read_write</code>             | local     | N/A                              |
| <code>others</code>                 | local     | N/A                              |
| <code>parameter_count</code>        | max/min   | N/A                              |
| <code>association_element_s</code>  | aggregate | <code>association_element</code> |

Use the `parameter_count` attribute to set a maximum or minimum number of elements for an association list. For example:

```
min parameter_count in association_list is 2
```

Use the `named_association` and the `positional_association` attributes to set named or positional associations on each element of an association list.

Use the `open` attribute to control the use of the open VHDL keyword in the association list. For example, the rule:

```
no open in association_list severity error
```

means that the open keyword is prohibited in association lists.

Use the `read_write` attribute to control the use of actual parameters in the association list. Consider the following rule:

```
no read_write in association_list severity error
```

This rule ensures that the same actual parameter is not assigned to formals of both mode in and mode out or inout.

Use the `association_element_s` attribute to point to templates of type `association_element`. This gives you extra control over individual elements in the list.

## Asynchronous Initialization Template

The `asynchronous_initialization` template is a primary template belonging to no classes. It contains the attributes shown in [Table 21](#).

**Table 21: asynchronous\_initialization Template**

| Attribute         | Kind     | Limit_Kind        |
|-------------------|----------|-------------------|
| identifier        | template | ID                |
| is_load           | local    | N/A               |
| is_reset          | local    | N/A               |
| is_set            | local    | N/A               |
| edge              | set      | N/A               |
| expression        | template | EXPRESSION        |
| object_definition | template | OBJECT_ITEM       |
| connectivity_path | template | connectivity_path |
| gated_in_unit     | template | ID                |
| cased_identifier  | template | ID                |

Leda recognizes all RTL reset expressions. However, if you want to limit the expression type, use the `expression` attribute.

Use the `object_definition` attribute to limit the type of the reset signal.



Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is asynchronous_initialization
 limit identifier to "^[a-z][a-z0-9_]*$"
end
 template T2 is asynchronous_initialization
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.



### Note

If you use a complex expression (containing more than one object) as a clock or reset, Leda automatically deactivates naming convention rules. This is because Leda assumes that the clock or reset is the output of a logic gate inferred by the complex expression, and not simply one of the objects in the expression.

## Attribute Declaration Template

The LRM (§4.4) defines the grammar for attribute declaration as follows:

```
attribute_declaration ::= attribute identifier : type_mark ;
```

The `attribute_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 22](#).

**Table 22: attribute\_declaration Template**

| Attribute                             | Kind     | Limit_Kind  |
|---------------------------------------|----------|-------------|
| <code>cased_identifier</code>         | template | ID          |
| <code>identifier</code>               | template | ID          |
| <code>type_mark</code>                | template | TYPE_MARK   |
| <code>declarative_region</code>       | template | REGION_PART |
| <code>one_declaration_per_line</code> | local    | N/A         |
| <code>comment</code>                  | local    | N/A         |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is attribute_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is attribute_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The attribute `identifier` is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Attribute Name Template

The LRM (§6.6) defines the grammar for attribute name as follows:

```
attribute_name ::=
 prefix [signature] ' attribute_designator [(expression)]
 attribute_designator ::= attribute_simple_name
```

The `attribute_name` template is a primary template belonging to the `EXPRESSION`, `NAME`, and `TARGET` classes. It contains the attributes shown in [Table 23](#).

**Table 23: attribute\_name Template**

| Attribute                         | Kind     | Limit_Kind            |
|-----------------------------------|----------|-----------------------|
| <code>name_prefix</code>          | template | EXPRESSION            |
| <code>named_entity_prefix</code>  | template | NAMED_ENTITY          |
| <code>expression</code>           | template | EXPRESSION            |
| <code>attribute_designator</code> | template | attribute_declaration |

The `named_entity_prefix` attribute ensures full compatibility with the LRM, something the `name_prefix` attribute does not do. Use the `named_entity_prefix` attribute to constrain prefixes that are types. For example:

```
integer' image(p)
```

## Attribute Specification Template

The LRM (§5.1) defines the grammar for attribute specification as follows:

```

attribute_specification ::= attribute attribute_designator of
 entity_specification is expression ;
entity_specification ::= entity_name_list : entity_class
entity_class ::=
 entity | architecture | configuration
 | procedure | function | package
 | type | subtype | constant
 | signal | variable | component
 | label | literal | units
 | group | file
entity_name_list ::=
 entity_designator { , entity_designator }
 | others
 | all
entity_designator ::= entity_tag [signature]
entity_tag ::= simple_name | character_literal | operator_symbol

```

The attribute\_specification template is a primary template belonging to no classes. It contains the attributes shown in [Table 24](#).

**Table 24: attribute\_specification Template**

| Attribute            | Kind     | Limit_Kind            |
|----------------------|----------|-----------------------|
| cased_label          | template | ID                    |
| label                | template | ID                    |
| attribute_designator | template | attribute_declaration |
| entity_designator    | template | NAME                  |
| expression           | template | EXPRESSION            |
| others               | local    | N/A                   |
| all                  | local    | N/A                   |
| entity               | local    | N/A                   |
| architecture         | local    | N/A                   |
| configuration        | local    | N/A                   |
| procedure            | local    | N/A                   |
| function             | local    | N/A                   |
| package              | local    | N/A                   |

**Table 24: attribute\_specification Template (Continued)**

| Attribute | Kind  | Limit_Kind |
|-----------|-------|------------|
| type      | local | N/A        |
| subtype   | local | N/A        |
| constant  | local | N/A        |
| signal    | local | N/A        |
| variable  | local | N/A        |
| component | local | N/A        |
| literal   | local | N/A        |
| units     | local | N/A        |
| group     | local | N/A        |
| file      | local | N/A        |

Use the `cased_label` just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is attribute_specification
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is attribute_specification
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Binary Operation Template

The `binary_operation` template is a primary template belonging to the `EXPRESSION` class. It contains the attributes shown in [Table 25](#).

**Table 25: `binary_operation` Template**

| Attribute                          | Kind     | Limit_Kind |
|------------------------------------|----------|------------|
| <code>left_expression</code>       | template | EXPRESSION |
| <code>right_expression</code>      | template | EXPRESSION |
| <code>type_mark</code>             | template | TYPE_MARK  |
| <code>operand_size_mismatch</code> | local    | N/A        |
| <code>operator_symbol</code>       | template | ID         |
| <code>value</code>                 | template | literal    |
| <code>evaluation_time</code>       | set      | N/A        |
| <code>operand_size_match</code>    | local    | N/A        |

Use the `operator_symbol` attribute with character strings or templates of class `ID` representing the acceptable operators. For example:

```
limit operator_symbol in binary_operation to "+"
```

You can set the `evaluation_time` attribute with any of these enumerated literals:

- `undefined`
- `locally_static_evaluation`
- `globally_static_evaluation`
- `dynamic_evaluation`

For example:

```
set evaluation_time to locally_static_evaluation
```

In addition to literal templates, you can use the `value` attribute for either integer values or character strings representing the literal value. For example:

```
limit value to 0
limit value to "0"
```

## Binding Indication Template

The LRM (§5.2.1) defines the grammar for binding indication as follows:

```
binding_indication ::=
 [use entity_aspect]
 [generic_map_aspect]
 [port_map_aspect]
entity_aspect ::=
 entity entity_name [(architecture_identifier)]
 | configuration configuration_name
 | open
generic_map_aspect ::=
 generic map (generic_association_list)
port_map_aspect ::=
 port map (port_association_list)
```

The `binding_indication` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 26](#).

**Table 26: binding\_indication Template**

| Attribute                       | Kind     | Limit_Kind       |
|---------------------------------|----------|------------------|
| <code>generic_map_aspect</code> | template | association_list |
| <code>port_map_aspect</code>    | template | association_list |
| <code>entity_aspect_name</code> | template | ID               |
| <code>entity</code>             | local    | N/A              |
| <code>configuration</code>      | local    | N/A              |

Use the `entity_aspect_name` attribute to configure the name of the entity you want to constrain. For example, to prohibit the use of “WORK” in the entity name, you could write the following rule:

```
template NO_WORK_IN_ENTITY_NAME is binding_indication
 limit entity_aspect_name to "^[^w][^o][^r][^k]"
end
template NO_WORK_IN_CONFIG_SPEC is configuration_specification
 limit binding_indication to NO_WORK_IN_CONFIG_SPEC
end
limit configuration_specification in all to NO_WORK_IN_CONFIG_SPEC
```

Use the `entity` and `configuration` attributes to force or prohibit entity or configuration entity aspects.

## Block Configuration Template

The LRM (§1.3.1) defines the grammar for block configuration as follows:

```

block_configuration ::=
 for block_specification
 { use_clause }
 { configuration_item }
 end for ;
configuration_item ::=
 block_configuration
 | component_configuration

```

The `block_configuration` template is a primary template belonging to the `REGION_PART` class. It contains the attributes shown in [Table 27](#).

**Table 27: block\_configuration Template**

| Attribute                            | Kind     | Limit_Kind                           |
|--------------------------------------|----------|--------------------------------------|
| <code>block_specification</code>     | template | <code>block_specification</code>     |
| <code>use_clause</code>              | template | <code>use_clause</code>              |
| <code>block_configuration</code>     | template | <code>block_configuration</code>     |
| <code>component_configuration</code> | template | <code>component_configuration</code> |

## Block Specification Template

The LRM (§1.3.1) defines the grammar for block specification as follows:

```

block_specification ::=
 architecture_name
 | block_statement_label
 | generate_statement_label [(index_specification)]

index_specification ::=
 discrete_range
 | static_expression

```

The `block_specification` template is a primary template belonging to the `REGION_PART` class. It contains the attributes shown in [Table 28](#).

**Table 28: `block_specification` Template**

| Attribute                             | Kind     | Limit_Kind |
|---------------------------------------|----------|------------|
| <code>index_specification</code>      | template | INDEX_SPEC |
| <code>architecture_name</code>        | local    | N/A        |
| <code>block_statement_label</code>    | local    | N/A        |
| <code>generate_statement_label</code> | local    | N/A        |

To limit the `index_specification` attribute, use a template that belongs to the `INDEX_SPEC` class. For example:

```

template RANGE_1 is range
 no null_range
 no downto
 set evaluation_time to locally_static_evaluation
end
limit index_specification in block_specification to RANGE_1
 severity error

```



## Block Statement Template

The LRM (§9.1) defines the grammar for block statement as follows:

```

block_statement ::=
 block_label :
 block [(guard_expression)] [is]
 block_header
 block_declarative_part
 begin
 block_statement_part
 end block [block_label] ;
block_header ::=
 [generic_clause [generic_map_aspect ;]]
 [port_clause [port_map_aspect ;]]
block_declarative_part ::=
 { block_declarative_item }
block_statement_part ::=
 { concurrent_statement }

```

The `block_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 29](#).

**Table 29: block\_statement Template**

| Attribute                                | Kind     | Limit_Kind                     |
|------------------------------------------|----------|--------------------------------|
| <code>cased_label</code>                 | template | ID                             |
| <code>label</code>                       | template | ID                             |
| <code>guard_expression</code>            | template | EXPRESSION                     |
| <code>generic_declaration</code>         | template | interface_constant_declaration |
| <code>generic_map_aspect</code>          | template | association_list               |
| <code>port_declaration</code>            | template | interface_signal_declaration   |
| <code>port_map_aspect</code>             | template | association_list               |
| <code>subprogram_declaration</code>      | template | subprogram_declaration         |
| <code>subprogram_body</code>             | template | subprogram_body                |
| <code>type_declaration</code>            | template | type_declaration               |
| <code>subtype_declaration</code>         | template | subtype_declaration            |
| <code>constant_declaration</code>        | template | constant_declaration           |
| <code>shared_variable_declaration</code> | template | shared_variable_declaration    |

**Table 29: block\_statement Template (Continued)**

| Attribute                         | Kind      | Limit_Kind                        |
|-----------------------------------|-----------|-----------------------------------|
| signal_declaration                | template  | signal_declaration                |
| file_declaration                  | template  | file_declaration                  |
| alias_declaration                 | template  | alias_declaration                 |
| attribute_declaration             | template  | attribute_declaration             |
| attribute_specification           | template  | attribute_specification           |
| disconnection_specification       | template  | disconnection_specification       |
| component_declaration             | template  | component_declaration             |
| configuration_specification       | template  | configuration_specification       |
| use_clause                        | template  | use_clause                        |
| group_template_declaration        | template  | group_template_declaration        |
| group_declaration                 | template  | group_declaration                 |
| assertion_statement               | template  | assertion_statement               |
| procedure_call_statement          | template  | procedure_call_statement          |
| process_statement                 | template  | process_statement                 |
| block_statement                   | template  | block_statement                   |
| selected_signal_assignment        | template  | selected_signal_assignment        |
| conditional_signal_assignment     | template  | conditional_signal_assignment     |
| component_instantiation_statement | template  | component_instantiation_statement |
| generate_statement                | template  | generate_statement                |
| declarative_region                | template  | REGION_PART                       |
| statement_format                  | template  | statement_format                  |
| statement_profile_s               | aggregate | CONCURRENT_STATEMENT              |
| declaration_profile_s             | aggregate | DECLARATIVE_ITEM                  |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is block_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is block_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Case Statement Template

The LRM (§8.8) defines the grammar for case statement as follows:

```
case_statement ::=
 [case_label :]
 case expression is
 case_statement_alternative
 {case_statement_alternative}
 end case[case_label] ;
case_statement_alternative ::= when choices => { sequence_of_statements }
```

The `case_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 30](#).

**Table 30: case\_statement Template**

| Attribute                                  | Kind     | Limit_Kind                                 |
|--------------------------------------------|----------|--------------------------------------------|
| <code>cased_label</code>                   | template | ID                                         |
| <code>label</code>                         | template | ID                                         |
| <code>expression</code>                    | template | EXPRESSION                                 |
| <code>choice</code>                        | template | EXPRESSION                                 |
| <code>assertion_statement</code>           | template | <code>assertion_statement</code>           |
| <code>report_statement</code>              | template | <code>report_statement</code>              |
| <code>procedure_call_statement</code>      | template | <code>procedure_call_statement</code>      |
| <code>if_statement</code>                  | template | <code>if_statement</code>                  |
| <code>signal_assignment_statement</code>   | template | <code>signal_assignment_statement</code>   |
| <code>variable_assignment_statement</code> | template | <code>variable_assignment_statement</code> |

**Table 30: case\_statement Template (Continued)**

| Attribute            | Kind      | Limit_Kind           |
|----------------------|-----------|----------------------|
| case_statement       | template  | case_statement       |
| loop_statement       | template  | loop_statement       |
| for_loop_statement   | template  | for_loop_statement   |
| while_loop_statement | template  | while_loop_statement |
| next_statement       | template  | next_statement       |
| exit_statement       | template  | exit_statement       |
| return_statement     | template  | return_statement     |
| wait_statement       | template  | wait_statement       |
| null_statement       | local     | N/A                  |
| others               | local     | N/A                  |
| declarative_region   | template  | REGION_PART          |
| statement_format     | template  | statement_format     |
| multiple_choices     | local     | N/A                  |
| alternative_count    | max/min   | N/A                  |
| statement_profile_s  | aggregate | SEQUENTIAL_STATEMENT |

Use the `multiple_choices` attribute to control the presence of more than one choice for the same alternative.

Use the `others` attribute to control the presence of the others alternative. Note that, if all alternatives are not explicit and there is no `others` clause, there is a VHDL syntax error.

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is case_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is case_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Clock Template

The clock template is a primary template belonging to no classes. It contains the attributes shown in [Table 31](#).

**Table 31: clock Template**

| Attribute         | Kind     | Limit_Kind        |
|-------------------|----------|-------------------|
| cased_identifier  | template | ID                |
| identifier        | template | ID                |
| expression        | template | EXPRESSION        |
| object_definition | template | OBJECT_ITEM       |
| edge              | set      | N/A               |
| connectivity_path | template | connectivity_path |
| starting_unit     | template | REGION_PART       |
| data              | local    | N/A               |
| gated_in_unit     | template | ID                |
| fixed_value       | local    | N/A               |

The clock template recognizes all RTL clock expressions. If you want to constrain the expression type for clocks, use the expression attribute. For example, you could use the expression attribute if the following code was not acceptable:

```
clk='1' and not clk'STABLE
```

Use the object\_definition attribute to limit the type of the clock signal.

Use the fixed\_value attribute to detect clocks with fixed values.

Use the edge attribute to control the active edge of the clock. You can set the edge attribute to any of the following values:

- Rising\_Edge
- Falling\_Edge
- Both\_Edges
- High\_Level
- Low\_Level

For example, the following VRSL code sets the active clock edge to rising:

```
template MY_CLOCK is clock
 set edge to rising_edge
end
```

Use the data attribute to constrain the use of the clock signal as data.

Use the gated\_in\_unit attribute to control the whereabouts of gated clocks with greater precision. Using this attribute, you can write post-elaboration rules that control gated clocks.

Use the cased\_identifier attribute just like the identifier attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is clock
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is clock
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The id1 identifier matches both templates, whereas the ID1 identifier only matches the first template. The attribute identifier is not case-sensitive because VHDL is not case-sensitive.



#### Note

If you use a complex expression (containing more than one object) as a clock or reset, Leda automatically deactivates the naming convention rules. This is because Leda assumes that the clock or reset is the output of a logic gate inferred by the complex expression and not simply one of the objects in the expression.

## Comment Template

The comment template is a primary template belonging to no classes. It contains the attributes shown in [Table 32](#).

**Table 32: comment Template**

| Attribute          | Kind     | Limit_Kind |
|--------------------|----------|------------|
| enclosing_filename | template | ID         |
| text               | template | ID         |

## Component Configuration Template

The LRM (§1.3.1) defines the grammar for component configuration as follows:

```

component_configuration ::=
 for component_specification
 [binding_indication ;]
 [block_configuration]
 end for;

```

The component configuration template is a primary template belonging to the REGION\_PART class. It contains the attributes shown in [Table 33](#).

**Table 33: component\_configuration Template**

| Attribute               | Kind     | Limit_Kind              |
|-------------------------|----------|-------------------------|
| component_specification | template | component_specification |
| binding_indication      | template | binding_indication      |
| block_configuration     | template | block_configuration     |

## Component Declaration Template

The LRM (§4.5) defines the grammar for component declaration as follows:

```

component_declaration ::=
 component identifier [is]
 [local_generic_clause]
 [local_port_clause]
 end component [component_simple_name]

```

The component\_declaration template is a primary template belonging to the OBJECT\_ITEM class. It contains the attributes shown in [Table 34](#).

**Table 34: component\_declaration Template**

| Attribute           | Kind     | Limit_Kind                     |
|---------------------|----------|--------------------------------|
| cased_identifier    | template | ID                             |
| identifier          | template | ID                             |
| generic_declaration | template | interface_constant_declaration |
| port_declaration    | template | interface_signal_declaration   |

**Table 34: component\_declaration Template (Continued)**

| Attribute          | Kind     | Limit_Kind  |
|--------------------|----------|-------------|
| declarative_region | template | REGION_PART |
| port_order         | local    | N/A         |
| port_count         | max/min  | N/A         |

Use the `port_order` attribute to define the order in which ports must appear. Specify the order according to the mode of the ports. You must specify the order using the following strings (otherwise Leda flags an error):

- in
- out
- inout
- buffer
- linkage

You can separate these strings using white space or commas, as shown in the following VRSL example, which uses white space to separate the specified “out inout in” port order:

```
limit port_order in entity_declaration to "out inout in"
message "ERROR: Ports are not in correct order"
severity ERROR
```

Using this example, if the specified port order is not followed or there is a port of mode buffer or linkage, the Leda Checker flags an error, and generates the specified message.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is component_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is component_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.



## Component Instantiation Statement Template

The LRM (§9.6) defines the grammar for component instantiation statement as follows:

```

component_instantiation_statement ::=
 instantiation_label :
 instantiated_unit
 [generic_map_aspect]
 [port_map_aspect] ;
instantiated_unit ::=
 [component] component_name
 | entity entity_name [(architecture_identifier)]
 | configuration configuration_name

```

The `component_instantiation_statement` template is a primary template belonging to the `CONCURRENT_STATEMENT` class. It contains the attributes shown in [Table 35](#).

**Table 35: component\_instantiation\_statement Template**

| Attribute                          | Kind     | Limit_Kind       |
|------------------------------------|----------|------------------|
| <code>cased_label</code>           | template | ID               |
| <code>label</code>                 | template | ID               |
| <code>unit_name</code>             | template | ID               |
| <code>generic_map_aspect</code>    | template | association_list |
| <code>port_map_aspect</code>       | template | association_list |
| <code>declarative_region</code>    | template | REGION_PART      |
| <code>entity</code>                | local    | N/A              |
| <code>configuration</code>         | local    | N/A              |
| <code>consistent_port_order</code> | local    | N/A              |
| <code>use_db_name</code>           | local    | N/A              |
| <code>db_instantiation</code>      | local    | N/A              |

- Use the `unit_name` attribute to configure the name of the instantiated component. For example, to ensure that the instantiated component's name corresponds to an existing entity, you could write the following rule:

```
limit unit_name in component_instantiation_statement to "^<entity>$"
```

- Use the `entity` and `configuration` attributes to force or forbid the presence of the `entity` and `configuration` VHDL keywords.

- Use the `use_db_name` attribute to ensure that the label name of the module instantiation does not collide with the name of a DB cell specified in the `$link_library` and found in a directory in `$search_path`.
- Use `db_instantiation` attribute to constrain the instantiation of db cells. This attribute can be used to ensure that the name of the module does not collide with the name of the cell in the `$link_library`. For example:

```
no db_instantiation in component_instantiation_statement

architecture A of E is

begin
 db_cell : comp(); // Fail -- use_db_name rule
 lab: db_cell(); // Fail -- no no db_instantiation rule

end A
```

- Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is component_instantiation_statement
 limit label to "^[a-z][a-z0-9_]*$"
end

template T2 is component_instantiation_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Component Specification Template

The LRM (§5.2) defines the grammar for component specification as follows:

```
component_specification ::= instantiation_list : component_name
```

The `component_specification` template is a primary template belonging to no classes. It contains the attributes shown in [Table 36](#).

**Table 36: component\_specification Template**

| Attribute                | Kind     | Limit_Kind |
|--------------------------|----------|------------|
| <code>cased_label</code> | template | ID         |
| <code>label</code>       | template | ID         |

**Table 36: component\_specification Template**

| Attribute | Kind  | Limit_Kind |
|-----------|-------|------------|
| others    | local | N/A        |
| all       | local | N/A        |
| postponed | local | N/A        |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is component_specification
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is component_specification
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `others` and `all` attributes to force or forbid the presence of the `others` and `all` VHDL keywords in an instantiation list.

## Concurrent Assertion Statement Template

The LRM (§9.4) defines the grammar for concurrent assertion statement as follows:

```

concurrent_assertion_statement ::= [label :] [postponed] assertion ;

```

The `concurrent_assertion_statement` template is a primary template belonging to the `CONCURRENT_STATEMENT` class. It contains the attributes shown in [Table 37](#).

**Table 37: concurrent\_assertion\_statement Template**

| Attribute                      | Kind     | Limit_Kind |
|--------------------------------|----------|------------|
| <code>cased_label</code>       | template | ID         |
| <code>label</code>             | template | ID         |
| <code>condition</code>         | template | EXPRESSION |
| <code>report_expression</code> | template | EXPRESSION |

**Table 37: concurrent\_assertion\_statement Template (Continued)**

| Attribute           | Kind     | Limit_Kind  |
|---------------------|----------|-------------|
| severity_expression | template | EXPRESSION  |
| declarative_region  | template | REGION_PART |
| postponed           | local    | N/A         |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is concurrent_assertion_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is concurrent_assertion_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Concurrent Procedure Call Statement Template

The LRM (§9.3) defines the grammar for concurrent procedure call as follows:

```

concurrent_procedure_call_statement ::= [label :] [postponed]
 procedure_call;

```

The `concurrent_procedure_call_statement` template is a primary template belonging to the `CONCURRENT_STATEMENT` class. It contains the attributes shown in [Table 38](#).

**Table 38: concurrent\_procedure\_call\_statement Template**

| Attribute                          | Kind     | Limit_Kind       |
|------------------------------------|----------|------------------|
| <code>cased_identifier</code>      | template | ID               |
| <code>cased_label</code>           | template | ID               |
| <code>label</code>                 | template | ID               |
| <code>identifier</code>            | template | ID               |
| <code>actual_parameter_part</code> | template | association_list |
| <code>declarative_region</code>    | template | REGION_PART      |
| <code>postponed</code>             | local    | N/A              |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is clock
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is clock
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Similarly, you use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is concurrent_procedure_call_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is concurrent_procedure_call_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Conditional Signal Assignment Template

The LRM (§9.5.1) defines the grammar for conditional signal assignment as follows:

```
conditional_signal_assignment ::=
 target <= options conditional_waveforms ;
```

The `conditional_signal_assignment` template is a primary template belonging to the `CONCURRENT_STATEMENT` class. It contains the attributes shown in [Table 39](#).

**Table 39: conditional\_signal\_assignment Template**

| Attribute                          | Kind     | Limit_Kind |
|------------------------------------|----------|------------|
| <code>cased_label</code>           | template | ID         |
| <code>label</code>                 | template | ID         |
| <code>target</code>                | template | TARGET     |
| <code>reject_expression</code>     | template | EXPRESSION |
| <code>operand_size_mismatch</code> | local    | N/A        |
| <code>postponed</code>             | local    | N/A        |

**Table 39: conditional\_signal\_assignment Template (Continued)**

| Attribute               | Kind      | Limit_Kind            |
|-------------------------|-----------|-----------------------|
| range_overflow          | local     | N/A                   |
| statement_format        | template  | statement_format      |
| transport               | local     | N/A                   |
| inertial                | local     | N/A                   |
| guarded                 | local     | N/A                   |
| read_write              | local     | N/A                   |
| waveform_count          | max/min   | N/A                   |
| conditional_waveforms_s | aggregate | conditional_waveforms |

Use the transport, inertial, and guarded attributes to force or forbid the presence of the corresponding VHDL keywords (transport, inertial, and guarded).

Use the waveform\_count attribute to set the maximum number of waveforms allowed.

Use the cased\_label attribute just like the label attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is conditional_signal_assignment
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is conditional_signal_assignment
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The lab1 label matches both templates, whereas the LAB1 label only matches the first template. The label attribute is not case-sensitive because VHDL is not case-sensitive.

## Conditional Waveforms Template

The LRM (§9.5.1) defines the grammar for conditional waveforms as follows:

```

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

```

This template is used in the conditional\_signal\_assignment template.

The conditional\_waveforms template is a secondary template belonging to no classes. It contains the attributes shown in [Table 40](#).

**Table 40: conditional\_waveforms Template**

| Attribute | Kind     | Limit_Kind |
|-----------|----------|------------|
| waveform  | template | waveform   |
| condition | template | EXPRESSION |

## Configuration Declaration Template

The LRM (§2.3) defines the grammar for configuration declaration as follows:

```

configuration_declaration ::=
 configuration identifier of entity_name is
 configuration_declarative_part
 block_configuration
 end [configuration] [configuration_simple_name] ;
configuration_declarative_part ::= { configuration_declarative_item }
configuration_declarative_item ::=
 use_clause | attribute_specification | group_declaration

```

The configuration\_declaration template is a primary template belonging to the REGION\_PART and OBJECT\_ITEM classes. It contains the attributes shown in [Table 41](#).

**Table 41: configuration\_declaration Template**

| Attribute               | Kind     | Limit_Kind                |
|-------------------------|----------|---------------------------|
| cased_identifier        | template | ID                        |
| identifier              | template | ID                        |
| use_clause              | template | use_clause                |
| attribute_specification | template | attribute_specification   |
| group_declaration       | template | group_declaration         |
| block_configuration     | template | block_configuration       |
| design_unit             | template | design_unit               |
| header_comment          | template | header_comment            |
| top_configuration       | template | configuration_declaration |
| file_name               | template | ID                        |
| ncs_file_name           | template | ID                        |
| file_layout             | template | file_layout               |
| line_count              | max/min  | N/A                       |
| file_length_count       | max/min  | N/A                       |
| related_unit            | local    | N/A                       |



Employ the `use_clause` attribute to constrain use-clauses declared inside a unit. To constrain use clauses (and library clauses) appearing before a unit, employ the `design_unit` template instead.

This template includes the following application-specific attributes that make it easier to write commonly-used rules:

- Use the `top_configuration` attribute to constrain the top-level configuration.  
To make this work, you must specify the name of the top-level architecture using the `-top` option of the command-line Checker.
- Use the `file_name` attribute to configure the name of the containing file. For example:

```
limit file_name in configuration_declaration to "config.vhd"
```

- Use the `line_count` attribute to specify the maximum or minimum lines of code allowed in a unit.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is configuration_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is configuration_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

- Use the `related_unit` attribute to check that the architecture or the configuration you can find in a file is related to the entity described in that same file.

## Configuration Specification Template

The LRM (§5.2) defines the grammar for configuration specification as follows:

```

configuration_specification ::=
 for component_specification binding_indication ;
component_specification ::= instantiation_list : component_name
instantiation_list ::=
 instantiation_label { , instantiation_label }
 | others
 | all

```

The configuration\_specification template is a primary template belonging to no classes. It contains the attributes shown in [Table 42](#).

**Table 42: configuration\_specification Template**

| Attribute             | Kind     | Limit_Kind         |
|-----------------------|----------|--------------------|
| binding_indication    | template | binding_indication |
| others                | local    | N/A                |
| all                   | local    | N/A                |
| consistent_port_order | local    | N/A                |

Use the others and all attributes to force or forbid the presence of the corresponding VHDL keywords (others and all).

## Constant Declaration Template

The LRM (§4.3.1.1) defines the grammar for constant declaration as follows:

```

constant_declaration ::=
 constant identifier_list : subtype_indication [:= expression] ;

```

The constant\_declaration template is a primary template belonging to the OBJECT\_ITEM class. It contains the attributes shown in [Table 43](#).

**Table 43: constant\_declaration Template**

| Attribute          | Kind     | Limit_Kind         |
|--------------------|----------|--------------------|
| cased_identifier   | template | ID                 |
| identifier         | template | ID                 |
| subtype_indication | template | subtype_indication |
| default            | template | EXPRESSION         |

**Table 43: constant\_declaration Template (Continued)**

| Attribute                | Kind     | Limit_Kind  |
|--------------------------|----------|-------------|
| declarative_region       | template | REGION_PART |
| deferred                 | local    | N/A         |
| one_declaration_per_line | local    | N/A         |
| comment                  | local    | N/A         |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```

template T1 is constant _declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is constant _declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Constrained Array Definition Template

The LRM (§3.2.1) defines the grammar for constrained array definition as follows:

```

constrained_array_definition ::=
 array index_constraint of element_subtype_indication
index_constraint ::= (discrete_range { , discrete_range })
discrete_range ::= discrete_subtype_indication | range

```

The `constrained_array_definition` template is a primary template belonging to no classes. It contains the attributes shown in [Table 44](#).

**Table 44: constrained\_array\_definition Template**

| Attribute          | Kind      | Limit_Kind         |
|--------------------|-----------|--------------------|
| subtype_indication | template  | subtype_indication |
| dimension_count    | max/min   | N/A                |
| index_constraint_s | aggregate | DISCRETE_RANGE     |

Use the `dimension_count` attribute to specify the maximum or minimum number of allowed dimensions. For example, the command:

```
max dimension_count in constrained_array_definition is 1
```

means that only 1-dimensional arrays are allowed.

To allow more than one dimension, set different constraints on the index expressions for each dimension. Make sure each aggregate points to a set of range templates.

## Design Unit Template

The LRM (§11.1) defines the grammar for design unit as follows:

```
design_unit ::= context_clause library_unit
context_clause ::= {context_item}
context_item ::= library_clause
 | use_clause
```

The `design_unit` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 45](#).

**Table 45: design\_unit Template**

| Attribute                   | Kind     | Limit_Kind              |
|-----------------------------|----------|-------------------------|
| <code>library_clause</code> | template | ID                      |
| <code>library_name</code>   | template | ID                      |
| <code>use_clause</code>     | template | <code>use_clause</code> |
| <code>use_work</code>       | local    | N/A                     |

Use the `library_clause` attribute to constrain the libraries imported into the design.

Use the `library_name` attribute to constrain the name of the library containing the unit.

## Disconnection Specification Template

The LRM (§5.3) defines the grammar for disconnection specifications as follows:

```

disconnection_specification ::=
 disconnect guarded_signal_specification after time_expression
guarded_signal_specification ::= guarded_signal_list : type_mark
signal_list ::=
 signal_name { ,signal_name }
 | others
 | all

```

The `disconnection_specification` template is a primary template belonging to no classes. It contains the attributes shown in [Table 46](#).

**Table 46: `disconnection_specification` Template**

| Attribute              | Kind     | Limit_Kind |
|------------------------|----------|------------|
| <code>type_mark</code> | template | TYPE_MARK  |
| <code>others</code>    | local    | N/A        |
| <code>all</code>       | local    | N/A        |

## Element Association Template

The LRM (§7.3.2) defines the grammar for element association as follows:

```

element_association ::= [choices =>] expression
choices ::= choice { | choice }
choice ::=
 simple_expression
 | discrete_range
 | element_simple_name
 | others

```

The `element_association` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 47](#).

**Table 47: `element_association` Template**

| Attribute                           | Kind     | Limit_Kind |
|-------------------------------------|----------|------------|
| <code>choice</code>                 | template | EXPRESSION |
| <code>positional_association</code> | local    | N/A        |
| <code>range_overflow</code>         | local    | N/A        |

**Table 47: element\_association Template (Continued)**

| Attribute         | Kind  | Limit_Kind |
|-------------------|-------|------------|
| named_association | local | N/A        |
| multiple_choices  | local | N/A        |
| others            | local | N/A        |

## Element Declaration Template

The LRM (§3.2.2) defines the grammar for element declaration as follows:

```
element_declaration ::= identifier_list : element_subtype_definition ;
```

The element\_declaration template is a secondary template belonging to no classes. It contains the attributes shown in [Table 48](#).

**Table 48: element\_declaration Template**

| Attribute          | Kind     | Limit_Kind         |
|--------------------|----------|--------------------|
| cased_identifier   | template | ID                 |
| identifier         | template | ID                 |
| subtype_indication | template | subtype_indication |

Use the cased\_identifier attribute just like the identifier attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is element_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is element_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The id1 identifier matches both templates, whereas the ID1 identifier only matches the first template. The identifier attribute is not case-sensitive because VHDL is not case-sensitive.

## Entity Declaration Template

The LRM (§1.1.1) defines the grammar for entity declaration as follows:

```
entity_declaration ::=
 entity identifier is
 entity_header
 entity_declarative_part
 [begin
 entity_statement_part]
end [entity] [entity_simple_name] ;
```

The LRM (§1.1.1) defines the grammar for entity header as follows:

```
entity_header ::=
 [formal_generic_clause]
 [formal_port_clause]
generic_clause ::=
 generic (generic_interface_list) ;
port_clause ::=
 port (port_interface_list) ;
```

The LRM (§1.1.2) defines the grammar for entity declarative part as follows:

```
entity_declarative_part ::=
 { entity_declarative_item }
entity_declarative_item ::=
 subprogram_declaration
 | subprogram_body
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | signal_declaration
 | shared variable_declaration
 | file_declaration
 | alias_declaration
 | attribute_declaration
 | attribute_specification
 | disconnection_specification
 | use_clause
 | group_template_declaration
 | group_declaration
```

The LRM (§1.1.3) defines the grammar for entity statement part as follows:

```
entity_statement_part ::=
 { entity_statement }
entity_statement ::=
 concurrent_assertion_statement
 | passive_concurrent_procedure_call
 | passive_process_stm
```

The `entity_declaration` template is a primary template belonging to the `REGION_PART` and `OBJECT_ITEM` classes. It contains the attributes shown in [Table 49](#).

**Table 49: `entity_declaration` Template**

| Attribute                                | Kind     | Limit_Kind                                  |
|------------------------------------------|----------|---------------------------------------------|
| <code>cased_identifier</code>            | template | ID                                          |
| <code>identifier</code>                  | template | ID                                          |
| <code>generic_declaration</code>         | template | <code>interface_constant_declaration</code> |
| <code>port_declaration</code>            | template | <code>interface_signal_declaration</code>   |
| <code>subprogram_declaration</code>      | template | <code>subprogram_declaration</code>         |
| <code>subprogram_body</code>             | template | <code>subprogram_body</code>                |
| <code>type_declaration</code>            | template | <code>type_declaration</code>               |
| <code>subtype_declaration</code>         | template | <code>subtype_declaration</code>            |
| <code>constant_declaration</code>        | template | <code>constant_declaration</code>           |
| <code>signal_declaration</code>          | template | <code>signal_declaration</code>             |
| <code>shared_variable_declaration</code> | template | <code>shared_variable_declaration</code>    |
| <code>use_db_name</code>                 | local    | N/A                                         |
| <code>file_declaration</code>            | template | <code>file_declaration</code>               |
| <code>alias_declaration</code>           | template | <code>alias_declaration</code>              |
| <code>attribute_declaration</code>       | template | <code>attribute_declaration</code>          |
| <code>attribute_specification</code>     | template | <code>attribute_specification</code>        |
| <code>disconnection_specification</code> | template | <code>disconnection_specification</code>    |
| <code>use_clause</code>                  | template | <code>use_clause</code>                     |
| <code>group_template_declaration</code>  | template | <code>group_template_declaration</code>     |
| <code>group_declaration</code>           | template | <code>group_declaration</code>              |
| <code>assertion_statement</code>         | template | <code>assertion_statement</code>            |
| <code>procedure_call_statement</code>    | template | <code>procedure_call_statement</code>       |
| <code>process_statement</code>           | template | <code>process_statement</code>              |
| <code>design_unit</code>                 | template | <code>design_unit</code>                    |



**Table 49: entity\_declaration Template (Continued)**

| Attribute             | Kind      | Limit_Kind         |
|-----------------------|-----------|--------------------|
| header_comment        | template  | header_comment     |
| top_entity            | template  | entity_declaration |
| file_name             | template  | ID                 |
| ncs_file_name         | template  | ID                 |
| file_layout           | template  | file_layout        |
| port_order            | local     | N/A                |
| port_count            | max/min   | N/A                |
| line_count            | max/min   | N/A                |
| declaration_profile_s | aggregate | DECLARATIVE_ITEM   |
| file_length           | max/min   | N/A                |

Employ the `use_clause` attribute to constrain the number of use-clauses declared inside the entity declaration (after the entity keyword). To constrain the number of use clauses (and library clauses) appearing before the entity keyword, use the `design_unit` template instead.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the `top_entity` attribute to constrain the top-level entity.

To make this work, you must specify the name of the top-level entity using the `-top` switch with the command-line Checker.

- Use the `file_name` attribute to configure the name of the containing file. For example:

```
limit file_name in entity_declaration to "<entity>.vhd"
```

- Use the `file_layout` attribute to control the organization of the containing file.
- Use the `port_order` attribute to specify the order in which ports must appear. Specify the order according to the mode of the ports using any of these strings: “in”, “out”, “inout”, “buffer”, and “linkage”. Any other strings cause Leda to flag an error. Separate the port strings using blanks or commas. For example:

```
limit port_order in entity_declaration to "out inout in"
message "ERROR: Ports are not in correct order" severity ERROR
```

In this example, if the order is not followed or a part of mode buffer or linkage appears, Leda flags an error and prints the specified message.

- Use the `use_db_name` attribute to ensure that the name of the module does not collide with the name of a DB cell specified in the `$link_library` and found in a directory in `$search_path`.
- Use the `line_count` attribute to specify the maximum or minimum lines of code allowed in a unit.
- Use the `declaration_profile_s` attribute to specify the order in which declarations must appear. For example:

```

limit declaration_profile_s in entity_declaration to
 first type_declaration
 then subtype_declaration
 then {}

```

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```

template T1 is entity_declaration
 limit identifier to "[a-z][a-z0-9_]*$"
end
template T2 is entity_declaration
 limit cased_identifier to "[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Enumeration Type Definition Template

The LRM (§3.1.1) defines the grammar for enumeration type definition as follows:

```
enumeration_type_definition ::=
 (enumeration_literal { , enumeration_literal })
enumeration_literal ::= identifier | character_literal
```

The enumeration\_type template is a primary template belonging to no classes. It contains the attributes shown in [Table 50](#).

**Table 50: enumeration\_type Template**

| Attribute         | Kind     | Limit_Kind |
|-------------------|----------|------------|
| cased_identifier  | template | ID         |
| identifier        | template | ID         |
| character_literal | local    | N/A        |
| element_count     | min_max  | N/A        |

For the command:

```
no character_literal in enumeration_type_definition
severity error
```

Leda flags an error for the following:

```
type BAD_ENUM is (NO, CHARACTER, LITERALS, ALLOWED, 'C');
```

but not for this:

```
type GOOD_ENUM is (NO, CHARACTER, LITERALS, ALLOWED);
```

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is enumeration_type_definition
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is enumeration_type_definition
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Exit Statement Template

The LRM (§8.11) defines the grammar for exit statement as follows:

```
exit_statement ::= [label :] exit [loop_label] [when condition] ;
```

The `exit_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 51](#).

**Table 51: exit\_statement Template**

| Attribute                       | Kind     | Limit_Kind  |
|---------------------------------|----------|-------------|
| <code>cased_label</code>        | template | ID          |
| <code>label</code>              | template | ID          |
| <code>loop_label</code>         | template | ID          |
| <code>condition</code>          | template | EXPRESSION  |
| <code>declarative_region</code> | template | REGION_PART |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is exit_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is exit_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Expression Template

The VRSL representation of expressions deviates from the LRM representation because the LRM is so detailed that it leads to long chains of design rules to express even simple expressions.

The expression template is a primary template belonging to the EXPRESSION class. It contains the attributes shown in [Table 52](#).

**Table 52: expression Template**

| Attribute       | Kind     | Limit_Kind |
|-----------------|----------|------------|
| operator_symbol | template | ID         |
| value           | template | literal    |
| operand         | template | NAME       |
| qualified       | local    | N/A        |
| type_conversion | local    | N/A        |
| evaluation_time | set      | N/A        |

Use the `operator_symbol` attribute to specify operators that are acceptable in expressions. You can specify character strings or templates of class ID that represent the acceptable operators. For example:

```
limit operator_symbol in expression to "+"
```

Use the `operand` attribute to write rules that constrain all operands in an expression.

Use the `evaluation_time` attribute to specify any of the following enumerated literals:

- `undefined`
- `locally_static_evaluation`
- `globally_static_evaluation`
- `dynamic_evaluation`

For example:

```
set evaluation_time to locally_static_evaluation
```

Use the `value` attribute to specify an integer or character string that represents the literal value. For example:

```
limit value to 0
limit value to "0"
```

Use the `type_conversion` attribute to control the types of conversion functions allowed, if any.

## FSM Template

Leda can infer finite state machines (FSMs) from a design and apply rules to constrain the type of FSM accepted. Leda can identify FSMs that use case statements to define the states and transitions. Leda cannot identify FSMs defined using if statements. Leda recognizes 1-process, 2-process, and 3-process FSM templates if all processes are in the same block (architecture or module).

The fsm template is a primary template belonging to no classes. It contains the attributes shown in [Table 53](#).

**Table 53: fsm Template**

| Attribute             | Kind     | Limit_Kind  |
|-----------------------|----------|-------------|
| block_count           | max/min  | N/A         |
| mealy                 | local    | N/A         |
| moore                 | local    | N/A         |
| state_count           | max/min  | N/A         |
| state_variable        | template | OBJECT_ITEM |
| transition_in_default | local    | N/A         |

Use the `state_count` attribute to constrain the number of states allowed in an FSM. Note that Leda only counts states that are reachable. Therefore, the number of states is not equal to the number of different values that the state variable can have. You can also use the `state_count` attribute to enforce that the number of states is a power of 2 (for example).

Use the `state_variable` attribute to constrain the properties of state variables (for example, the name and type).

Use the `block_count` attribute to constrain the number of processes used to infer an FSM (for example, 1-, 2-, or 3-block FSMs).

Use the `moore` attribute to specify that only Moore-style FSMs are allowed, as shown in the following example:

```
force moore in fsm
```

Use the `mealy` attribute to specify that only Mealy-style FSMs are allowed, as shown in the following example:

```
force mealy in fsm
```

Use the `transition_in_default` attribute to force the use of a default clause in the case statement (even if one isn't necessary).

You can find a set of prepackaged FSM rules in the `STATE_MACHINES` ruleset of the Leda general coding guidelines policy. For more information, see the [Leda General Coding Rules Guide](#).

## File Declaration Template

The LRM (§4.3.1.4) defines the grammar for file declaration as follows:

```
file_declaration ::= file identifier_list : subtype_indication
 [file_open_information] ;
file_open_information ::= [open file_open_kind_expression] is
 file_logical_name
file_logical_name ::= string_expression
```

The `file_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 54](#).

**Table 54: file\_declaration Template**

| Attribute                             | Kind     | Limit_Kind                      |
|---------------------------------------|----------|---------------------------------|
| <code>cased_identifier</code>         | template | ID                              |
| <code>identifier</code>               | template | ID                              |
| <code>subtype_indication</code>       | template | <code>subtype_indication</code> |
| <code>logical_name</code>             | template | <code>simple_name</code>        |
| <code>open_kind</code>                | template | <code>simple_name</code>        |
| <code>declarative_region</code>       | template | REGION_PART                     |
| <code>one_declaration_per_line</code> | local    | N/A                             |
| <code>comment</code>                  | local    | N/A                             |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is file_declaration
 limit identifier to "^[a-z] [a-z0-9_]*$"
end
template T2 is file_declaration
 limit cased_identifier to "^[a-z] [a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## File Layout Template

Use the `file_layout` template to control the layout of the file containing your VHDL code. Note that you can also set some of these same things using the VHDL Beautifier that is part of the Checker tool. For more information on using the VHDL Beautifier, see the [Leda User Guide](#).

The `file_layout` template is a primary template belonging to the `no` classes. It contains the attributes shown in [Table 55](#).

**Table 55: file\_layout Template**

| Attribute                        | Kind     | Limit_Kind                  |
|----------------------------------|----------|-----------------------------|
| <code>characters_per_line</code> | max/min  | N/A                         |
| <code>cased_identifier</code>    | template | ID                          |
| <code>header_comment</code>      | template | <code>header_comment</code> |
| <code>identifier</code>          | template | ID                          |
| <code>unit_count</code>          | max/min  | N/A                         |
| <code>file_length</code>         | max/min  | N/A                         |
| <code>related_units</code>       | local    | N/A                         |

Use the `header_comment` attribute to control the appearance of header comments at the start of the file.

Use the `unit_count` attribute to control the number of units allowed in a file.



Use the `file_length` attribute to control the number of lines allowed in a file.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is file_layout
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is file_layout
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `characters_per_line` attribute to allow the number of characters on each line to be constrained.

## File Type Definition Template

The LRM (§3.4) defines the grammar for file type definition as follows:

```
file_type_definition ::= file of type_mark
```

The `file_type_definition` template is a primary template belonging to no classes. It contains the attribute shown in [Table 56](#).

**Table 56: file\_type\_definition Template**

| Attribute              | Kind     | Limit_Kind |
|------------------------|----------|------------|
| <code>type_mark</code> | template | TYPE_MARK  |

## Flipflop Template

The flipflop template is a primary template belonging to no classes. It contains the attributes shown in [Table 57](#).

**Table 57: flipflop Template**

| Attribute                                | Kind     | Limit_Kind                               |
|------------------------------------------|----------|------------------------------------------|
| <code>asynchronous_initialization</code> | limit    | <code>asynchronous_initialization</code> |
| <code>cased_identifier</code>            | template | ID                                       |
| <code>identifier</code>                  | template | ID                                       |
| <code>input</code>                       | template | EXPRESSION                               |

**Table 57: flipflop Template (Continued)**

| Attribute                  | Kind     | Limit_Kind                 |
|----------------------------|----------|----------------------------|
| output                     | template | EXPRESSION                 |
| synchronous_initialization | limit    | synchronous_initialization |
| clock                      | template | clock                      |
| data_signal                | template | data_signal                |
| has_clock_as_data          | local    | N/A                        |

Use the identifier attribute to control the names of flipflops inferred in the code. You can also do this using the output attribute.

Use the input attribute to control the expressions driving a flipflop.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```

template T1 is flipflop
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is flipflop
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Floating Type Definition Template

The LRM (§3.1.4) defines the grammar for floating point type as follows:

```
floating_point_type ::= range_constraint
```

The `floating_type_definition` template is a primary template belonging to no classes. It contains the attribute shown in [Table 58](#).

**Table 58: floating\_type\_definition Template**

| Attribute | Kind     | Limit_Kind |
|-----------|----------|------------|
| range     | template | range      |

## For Loop Statement Template

The LRM (§8.9) defines the grammar for loop statement as follows:

```

loop_statement ::=
 [loop_label :]
 [iteration_scheme] loop
 sequence_of_statements
 end loop [loop_label] ;
iteration_scheme ::=
 while condition
 | for loop_parameter_specification
parameter_specification ::=
 identifier in discrete_range

```

The `for_loop_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 59](#).

**Table 59: for\_loop\_statement Template**

| Attribute                                  | Kind     | Limit_Kind                                 |
|--------------------------------------------|----------|--------------------------------------------|
| <code>cased_label</code>                   | template | ID                                         |
| <code>label</code>                         | template | ID                                         |
| <code>subtype_indication</code>            | template | <code>subtype_indication</code>            |
| <code>assertion_statement</code>           | template | <code>assertion_statement</code>           |
| <code>report_statement</code>              | template | <code>report_statement</code>              |
| <code>procedure_call_statement</code>      | template | <code>procedure_call_statement</code>      |
| <code>if_statement</code>                  | template | <code>if_statement</code>                  |
| <code>signal_assignment_statement</code>   | template | <code>signal_assignment_statement</code>   |
| <code>variable_assignment_statement</code> | template | <code>variable_assignment_statement</code> |
| <code>case_statement</code>                | template | <code>case_statement</code>                |
| <code>loop_statement</code>                | template | <code>loop_statement</code>                |
| <code>for_loop_statement</code>            | template | <code>for_loop_statement</code>            |
| <code>while_loop_statement</code>          | template | <code>while_loop_statement</code>          |
| <code>next_statement</code>                | template | <code>next_statement</code>                |
| <code>exit_statement</code>                | template | <code>exit_statement</code>                |
| <code>return_statement</code>              | template | <code>return_statement</code>              |

**Table 59: for\_loop\_statement Template (Continued)**

| Attribute           | Kind      | Limit_Kind           |
|---------------------|-----------|----------------------|
| wait_statement      | template  | wait_statement       |
| null_statement      | local     | N/A                  |
| declarative_region  | template  | REGION_PART          |
| statement_format    | template  | statement_format     |
| statement_profile_s | aggregate | SEQUENTIAL_STATEMENT |
| null_statement      | local     | N/A                  |
| statement_profile_s | aggregate | SEQUENTIAL_STATEMENT |
| evaluation_time     | set       | N/A                  |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is for_loop_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is for_loop_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Formal Parameter Template

The LRM (§2.1) defines the grammar for formal parameter as follows:

```
formal_parameter_list ::= parameter_interface_list
```

The formal\_parameter template is a primary template belonging to the OBJECT\_ITEM class. It contains the attributes shown in [Table 60](#).

**Table 60: formal\_parameter Template**

| Attribute                | Kind     | Limit_Kind         |
|--------------------------|----------|--------------------|
| cased_identifier         | template | ID                 |
| identifier               | template | ID                 |
| subtype_indication       | template | subtype_indication |
| default                  | template | EXPRESSION         |
| in                       | local    | N/A                |
| out                      | local    | N/A                |
| inout                    | local    | N/A                |
| constant                 | local    | N/A                |
| one_declaration_per_line | local    | N/A                |
| signal                   | local    | N/A                |
| comment                  | local    | N/A                |
| variable                 | local    | N/A                |
| file                     | local    | N/A                |

The in, out, and inout attributes correspond to the different modes allowed for formal subprogram parameters. For example, the following command:

```
no out in formal_parameter severity error
```

means that parameters of mode “out” are not allowed.

The constant, signal, variable, and file attributes correspond to the different classes allowed for formal subprogram parameters. For example, the following command:

```
force signal in formal_parameter severity error
```

means that only parameters of class “signal” are allowed.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is formal_parameter
 limit identifier to "[a-z][a-z0-9_]*$"
end
template T2 is formal_parameter
 limit cased_identifier to "[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Function Call Template

The LRM (§7.3.3) defines the grammar for function call as follows:

```
function_call ::= function_name [(actual_parameter_part)]
actual_parameter_part ::= parameter_association_list
```

The `function_call` template is a primary template belonging to the `EXPRESSION` class. It contains the attributes shown in [Table 61](#).

**Table 61: function\_call Template**

| Attribute                          | Kind     | Limit_Kind       |
|------------------------------------|----------|------------------|
| <code>cased_identifier</code>      | template | ID               |
| <code>identifier</code>            | template | ID               |
| <code>actual_parameter_part</code> | template | association_list |

Use the `identifier` attribute to control calls to functions outside the current unit. For example, to signal an error if the function `RISING_EDGE` in the package `STD_LOGIC_1164` is called, you could write this rule as follows:

```
template BAD_FUNC_NAME is identifier
 limit error_id to "IEEE.STD_LOGIC_1164.RISING_EDGE"
end
limit identifier in function_call to BAD_FUNC_NAME
 message "Illegal function call" severity error
```

With this rule activated, Leda flags an error if the `error_id` attribute matches.

Use the `actual_parameter_part` attribute to point to association list templates that enable different constraints to be put on the parameters. For example, to impose named association on parameter lists unless there are only two or less parameters, you could write this rule as follows:

```
template AP1 is association_list
 min parameter_count is 2
end
limit actual_parameter_part in function_call to AP1 severity note
 if AP1 then
 force named_association in association_list
 message "Named association must be used here"
 severity error
 end if
```

Because it is not an error if AP1 is not matched, this example uses a severity of “note” to catch the mismatches. Leda then continues testing.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is function_call
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is function_call
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Generate Statement Template

The LRM (§9.7) defines the grammar for generate statement as follows:

```

generate_statement ::=
 generate_label :
 generation_scheme generate
 [{ block_declarative_item }
 begin]
 { concurrent_statement }
 end generate [generate_label] ;
generation_scheme ::=
 for generate_parameter_specification
 | if condition
parameter_specification ::=
 identifier in discrete_range

```

The generate\_statement template is a primary template belonging to the CONCURRENT\_STATEMENT class. It contains the attributes shown in [Table 62](#).

**Table 62: generate\_statement Template**

| Attribute                   | Kind     | Limit_Kind                  |
|-----------------------------|----------|-----------------------------|
| cased_label                 | template | ID                          |
| label                       | template | ID                          |
| discrete_range              | template | range                       |
| condition                   | template | EXPRESSION                  |
| subprogram_declaration      | template | subprogram_declaration      |
| subprogram_body             | template | subprogram_body             |
| type_declaration            | template | type_declaration            |
| subtype_declaration         | template | subtype_declaration         |
| constant_declaration        | template | constant_declaration        |
| shared_variable_declaration | template | shared_variable_declaration |
| signal_declaration          | template | signal_declaration          |
| file_declaration            | template | file_declaration            |
| alias_declaration           | template | alias_declaration           |
| component_declaration       | template | component_declaration       |
| attribute_declaration       | template | attribute_declaration       |



**Table 62: generate\_statement Template (Continued)**

| Attribute                         | Kind     | Limit_Kind                        |
|-----------------------------------|----------|-----------------------------------|
| attribute_specification           | template | attribute_specification           |
| disconnection_specification       | template | disconnection_specification       |
| configuration_specification       | template | configuration_specification       |
| use_clause                        | template | use_clause                        |
| group_template_declaration        | template | group_template_declaration        |
| group_declaration                 | template | group_declaration                 |
| assertion_statement               | template | assertion_statement               |
| procedure_call_statement          | template | procedure_call_statement          |
| process_statement                 | template | process_statement                 |
| block_statement                   | template | block_statement                   |
| selected_signal_assignment        | template | selected_signal_assignment        |
| conditional_signal_assignment     | template | conditional_signal_assignment     |
| component_instantiation_statement | template | component_instantiation_statement |
| generate_statement                | template | generate_statement                |
| declarative_region                | template | REGION_PART                       |

Use the condition attribute with if\_generate statements and the discrete\_range attribute with for\_generate statements.

Use the cased\_label attribute just like the label attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is generate_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is generate_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The lab1 label matches both templates, whereas the LAB1 label only matches the first template. The label attribute is not case-sensitive because VHDL is not case-sensitive.

## Generic Declaration Template

The `generic_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 63](#).

**Table 63: generic\_declaration Template**

| Attribute                             | Kind     | Limit_Kind         |
|---------------------------------------|----------|--------------------|
| <code>cased_identifier</code>         | template | ID                 |
| <code>identifier</code>               | template | ID                 |
| <code>subtype_indication</code>       | template | subtype_indication |
| <code>default</code>                  | template | EXPRESSION         |
| <code>declarative_region</code>       | template | REGION_PART        |
| <code>constant</code>                 | local    | N/A                |
| <code>in</code>                       | local    | N/A                |
| <code>one_declaration_per_line</code> | local    | N/A                |
| <code>comment</code>                  | local    | N/A                |

Use the “constant” and “in” attributes to control the presence of the constant and in VHDL keywords. For example, the following command:

```
force in in generic_declaration
```

means that the “in” keyword must be present for generic declarations.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is generic_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is generic_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Group Declarations

The LRM (§4.7) defines the grammar for group declaration as follows:

```
group_declaration ::=
group identifier : group_template_name (group_constituent_list) ;
group_constituent_list ::= group_constituent { , group_constituent }
group_constituent ::= name | character_literal
```

You cannot limit a `group_declaration` using the current version of Leda.

## Group Template Declarations

The LRM (§4.6) defines the grammar for group template declaration as follows:

```
group_template_declaration ::=
 group identifier is (entity_class_entry_list)
entity_class_entry_list ::=
 entity_class_entry { , entity_class_entry }
entity_class_entry ::= entity_class [<>]
```

You cannot limit a `group_template_declaration` using the current version of Leda.

## Header Comment Template

Use the `header_comment` template to control the comments that come before a VHDL unit or subprogram and make sure that each field is present. The `header_comment` template is a secondary template belonging to no classes. It contains the attribute shown in [Table 64](#).

**Table 64: header\_comment Template**

| Attribute                 | Kind     | Limit_Kind |
|---------------------------|----------|------------|
| <code>comment_line</code> | template | ID         |

Use the `comment_line` attribute to specify fields that must be present in a header comment. Consider the following template:

```
--
-- unit : architecture A of E
-- author : Me
-- date : November 1998
--
template UNIT is header_comment
 limit comment_line to "^-- unit :"
end
template AUTHOR is header_comment
 limit comment_line to "^-- author :"
end
template DATE is header_comment
 limit comment_line to "^-- date:"
end

limit header_comment in entity_declaration to allof
 UNIT message "Unit field is missing from header comment",
 AUTHOR message "Author field is missing from header comment",
 DATE message "Date field is missing from header comment"
```



### Note

Observe the use of the `allof` keyword to associate several messages with the same command.

## Identifier Template

The LRM (§13.3) defines the grammar for identifier as follows:

```
identifier ::= basic_identifier | extended_identifier
```

The identifier template is a primary template belonging to the ID class. It contains the attributes shown in the [Table 65](#).

**Table 65: identifier Template**

| Attribute       | Kind     | Limit_Kind |
|-----------------|----------|------------|
| limit_id        | template | ID         |
| error_id        | template | ID         |
| character_count | max/min  | N/A        |

Use character strings with the limit\_id attribute to specify the acceptable value, as shown in the following example:

```
limit limit_id to "^STD_LOGIC_1164$"
```

Use the error\_id attribute to indicate character strings that cannot be present. For example, Leda flags an error for the following rule when a signal declaration has the name BAD\_SIG\_NAME.

```
template BAD_SIG_ID is identifier
 limit error_id to "BAD_SIG_NAME"
end
limit identifier in signal_declaration to BAD_SIG_ID
 severity error
```

Use the character\_count attribute to specify the maximum or minimum number of characters allowed in an identifier.

For more information on writing naming convention rules, see [“Parameterizing Rules Using Regular Expressions”](#) on page 28.

## If Statement Template

The LRM (§8.7) defines the grammar for if statement as follows:

```

if_statement ::=
 [if_label :]
 if condition then
 sequence_of_statements
 { elsif condition then
 sequence_of_statements }
 [else
 sequence_of_statements }
 end if [if_label] ;

```

The if\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 66](#).

**Table 66: if\_statement Template**

| Attribute                     | Kind     | Limit_Kind                    |
|-------------------------------|----------|-------------------------------|
| cased_label                   | template | ID                            |
| label                         | template | ID                            |
| assertion_statement           | template | assertion_statement           |
| report_statement              | template | report_statement              |
| procedure_call_statement      | template | procedure_call_statement      |
| if_statement                  | template | if_statement                  |
| indentation_depth             | max/min  | N/A                           |
| signal_assignment_statement   | template | signal_assignment_statement   |
| variable_assignment_statement | template | variable_assignment_statement |
| case_statement                | template | case_statement                |
| loop_statement                | template | loop_statement                |
| for_loop_statement            | template | for_loop_statement            |
| while_loop_statement          | template | while_loop_statement          |
| next_statement                | template | next_statement                |
| exit_statement                | template | exit_statement                |
| return_statement              | template | return_statement              |
| wait_statement                | template | wait_statement                |

**Table 66: if\_statement Template (Continued)**

| Attribute           | Kind      | Limit_Kind           |
|---------------------|-----------|----------------------|
| null_statement      | local     | N/A                  |
| else                | local     | N/A                  |
| declarative_region  | template  | REGION_PART          |
| statement_format    | template  | statement_format     |
| condition_count     | max/min   | N/A                  |
| condition_s         | aggregate | EXPRESSION           |
| statement_profile_s | aggregate | SEQUENTIAL_STATEMENT |

You can constrain all sequential statements and the statement profile within an if statement. Use the else clause to force or forbid the use of the else alternative.

Use the condition\_count attribute to constrain the number of conditions in one if statement. For example:

```
if then
elseif ... then
else ... then
end if
```

In this example, there are three conditions in a single if statement. There are no nested if statements. So, the following rule is violated by this example:

```
max condition_count in if_statement is 2
```

Use the indentation\_depth attribute to constrain the number of nested if's in an if statement. For example:

```
if ... then
 if ... then
 if ... then
 end if
 end if
 end if
end if
```

In this example, there are three nested if's. Note that the current if is also counted. So, the following rule is violated by this example:

```
max indentation_depth in if_statement is 2
```

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is if_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is if_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Indexed Name Template

The LRM (§6.4) defines the grammar for indexed name as follows:

```
indexed_name ::= prefix (expression { , expression })
```

The `indexed_name` template is a secondary template belonging to the `EXPRESSION`, `NAME`, and `TARGET` classes. It contains the attributes shown in [Table 67](#).

**Table 67: indexed\_name Template**

| Attribute                      | Kind     | Limit_Kind  |
|--------------------------------|----------|-------------|
| <code>cased_identifier</code>  | template | ID          |
| <code>identifier</code>        | template | ID          |
| <code>object_definition</code> | template | OBJECT_ITEM |
| <code>range_overflow</code>    | local    | N/A         |
| <code>expression</code>        | template | EXPRESSION  |
| <code>flipflop</code>          | template | flipflop    |
| <code>latch</code>             | template | latch       |

Use the `object_definition` attribute to control the type of definition associated with a `simple_name`. For example, to consider just the ports in a given expression, you could write the following rule:

```
template CLK_NAME is simple_name
 limit object_definition to port_declaration
end
```



There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the flipflop attribute to control whether or not the object infers a flipflop
- Use the latch attribute to control whether or not the object infers a latch

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is indexed_name
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is indexed_name
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Integer Type Definition Template

The LRM (§3.1.2) defines the grammar for integer type definition as follows:

```
integer_type_definition ::= range_constraint
```

The `integer_type_definition` template is a primary template belonging to no classes. It contains the attribute shown in [Table 68](#).

**Table 68: integer\_type\_definition Template**

| Attribute | Kind     | Limit_Kind |
|-----------|----------|------------|
| range     | template | range      |

## Interface File Declaration Template

The LRM (§4.3.2) defines the grammar for interface file declaration as follows:

```
interface_file_declaration ::= file identifier_list subtype_indication
```

The `interface_file_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 69](#).

**Table 69: interface\_file\_declaration Template**

| Attribute                             | Kind     | Limit_Kind                      |
|---------------------------------------|----------|---------------------------------|
| <code>cased_identifier</code>         | template | ID                              |
| <code>identifier</code>               | template | ID                              |
| <code>subtype_indication</code>       | template | <code>subtype_indication</code> |
| <code>declarative_region</code>       | template | REGION_PART                     |
| <code>file</code>                     | local    | N/A                             |
| <code>one_declaration_per_line</code> | local    | N/A                             |
| <code>comment</code>                  | local    | N/A                             |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the `identifier` is case-sensitive. Consider the following templates:

```
template T1 is interface_file_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is interface_file_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Interface Variable Declaration Template

The LRM (§4.3.2) defines the grammar for interface variable declaration as follows:

```
interface_variable_declaration ::=
 [variable] identifier_list : [mode] subtype_indication
 [:= static_expression]
mode ::= in | out | inout | buffer | linkage
```

The `interface_variable_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 70](#).

**Table 70: interface\_variable\_declaration Template**

| Attribute                             | Kind     | Limit_Kind         |
|---------------------------------------|----------|--------------------|
| <code>cased_identifier</code>         | template | ID                 |
| <code>identifier</code>               | template | ID                 |
| <code>subtype_indication</code>       | template | subtype_indication |
| <code>default</code>                  | template | EXPRESSION         |
| <code>declarative_region</code>       | template | REGION_PART        |
| <code>variable</code>                 | local    | N/A                |
| <code>in</code>                       | local    | N/A                |
| <code>out</code>                      | local    | N/A                |
| <code>inout</code>                    | local    | N/A                |
| <code>buffer</code>                   | local    | N/A                |
| <code>linkage</code>                  | local    | N/A                |
| <code>one_declaration_per_line</code> | local    | N/A                |
| <code>comment</code>                  | local    | N/A                |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the `identifier` is case-sensitive. Consider the following templates:

```
template T1 is interface_variable_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is interface_variable_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The identifier attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Latch Template

The latch template is a primary template belonging to no classes. It contains the attributes shown in [Table 71](#).

**Table 71: latch Template**

| Attribute                                | Kind     | Limit_Kind                               |
|------------------------------------------|----------|------------------------------------------|
| <code>asynchronous_initialization</code> | limit    | <code>asynchronous_initialization</code> |
| <code>cased_identifier</code>            | template | ID                                       |
| <code>identifier</code>                  | template | ID                                       |
| <code>tinput</code>                      | template | EXPRESSION                               |
| <code>output</code>                      | template | EXPRESSION                               |
| <code>synchronous_initialization</code>  | limit    | <code>synchronous_initialization</code>  |
| <code>clock</code>                       | template | clock                                    |
| <code>data_signal</code>                 | template | <code>data_signal</code>                 |
| <code>has_clock_as_data</code>           | local    | N/A                                      |

Use the `identifier` attribute to control the names of latches inferred in the code. Note that you can accomplish the same thing using the `output` attribute. For example, to ensure that latches all have an identifier with the suffix “`_latch`”, you could define the following template:

```
template LATCH_ID is latch
 limit identifier to "_latch$"
end
```

Use the `input` attribute to control expressions driving a latch. You can point this attribute to any template of the `EXPRESSION` class.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is latch
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is latch
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Literal Template

The LRM (§7.3.1) defines the grammar for literal template as follows:

```
literal ::= numeric_literal
 | enumeration_literal
 | string_literal
 | bit_string_literal
 | null
numeric_literal ::= abstract_literal | physical_literal
```

The literal template is a primary template belonging to the `EXPRESSION` class. It contains the attributes shown in [Table 72](#).

**Table 72: literal Template**

| Attribute                    | Kind     | Limit_Kind |
|------------------------------|----------|------------|
| <code>value</code>           | template | literal    |
| <code>null</code>            | local    | N/A        |
| <code>use_exponent</code>    | local    | N/A        |
| <code>base</code>            | set      | N/A        |
| <code>base_specifier</code>  | set      | N/A        |
| <code>value_type</code>      | set      | N/A        |
| <code>evaluation_time</code> | set      | N/A        |

You can use integer values or character strings with the `value` attribute, as shown in the following examples:

```
limit value in literal to 0
limit value in literal to "0"
```

In the first example, the integer value zero is represented, whereas in the second case, the enumerated literal “0” or the bit\_string “0” is represented. The actual interpretation depends on the `value_type` attribute.

You can set the `value_type` attribute to any of the following enumerated literals:

- `any_literal_type`
- `abstract_literal_type`
- `physical_literal_type`
- `enumerated_literal_type`
- `string_literal_type`
- `bit_string_literal_type`

You can set the `evaluation_time` attribute to any of the following enumerated literals:

- `undefined`
- `locally_static_evaluation`
- `globally_static_evaluation`
- `dynamic_evaluation`

For example, the following rule sets the `evaluation_time` attribute to the `locally_static_evaluation` enumerated literal.

```
set evaluation_time to locally_static_evaluation
```

## Loop Statement Template

The LRM (§8.9) defines the grammar for loop statement as follows:

```

loop_statement ::=
 [loop_label :]
 [iteration_scheme] loop
 sequence_of_statements
 end loop [loop_label] ;
iteration_scheme ::=
 while condition
 | for loop_parameter_specification
parameter_specification ::=
 identifier in discrete_range

```

The loop\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 73](#).

**Table 73: loop\_statement Template**

| Attribute                     | Kind     | Limit_Kind                    |
|-------------------------------|----------|-------------------------------|
| cased_label                   | template | ID                            |
| label                         | template | ID                            |
| assertion_statement           | template | assertion_statement           |
| report_statement              | template | report_statement              |
| procedure_call_statement      | template | procedure_call_statement      |
| if_statement                  | template | if_statement                  |
| signal_assignment_statement   | template | signal_assignment_statement   |
| variable_assignment_statement | template | variable_assignment_statement |
| case_statement                | template | case_statement                |
| loop_statement                | template | loop_statement                |
| for_loop_statement            | template | for_loop_statement            |
| while_loop_statement          | template | while_loop_statement          |
| next_statement                | template | next_statement                |
| exit_statement                | template | exit_statement                |
| return_statement              | template | return_statement              |
| wait_statement                | template | wait_statement                |

**Table 73: loop\_statement Template (Continued)**

| Attribute           | Kind      | Limit_Kind           |
|---------------------|-----------|----------------------|
| null_statement      | local     | N/A                  |
| declarative_region  | template  | REGION_PART          |
| statement_format    | template  | statement_format     |
| statement_profile_s | aggregate | SEQUENTIAL_STATEMENT |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is loop_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is loop_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.



## Next Statement Template

The LRM (§8.10) defines the grammar for next statement as follows:

```
next_statement ::= [label :] next [loop_label] [when condition] ;
```

The next\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 74](#).

**Table 74: next\_statement Template**

| Attribute          | Kind     | Limit_Kind  |
|--------------------|----------|-------------|
| cased_label        | template | ID          |
| label              | template | ID          |
| loop_label         | template | ID          |
| condition          | template | EXPRESSION  |
| declarative_region | template | REGION_PART |

Use the cased\_label attribute just like the label attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is loop_statement
 limit label to "[a-z][a-z0-9_]*$"
end
template T2 is loop_statement
 limit cased_label to "[a-z][a-z0-9_]*$"
end
```

The lab1 label matches both templates, whereas the LAB1 label only matches the first template. The label attribute is not case-sensitive because VHDL is not case-sensitive.

## Package Body Template

The LRM (§2.6) defines the grammar for package body as follows:

```

package_body ::=
 package body package_simple_name is
 package_body_declarative_part
 end [package body] [package_simple_name] ;

package_body_declarative_part ::=
 { package_body_declarative_item }

package_body_declarative_item ::=
 subprogram_declaration
 | subprogram_body
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | shared_variable_declaration
 | file_declaration
 | alias_declaration
 | use_clause
 | group_template_declaration
 | group declaration

```

The package\_body template is a primary template belonging to the REGION\_PART class. It contains the attributes shown in [Table 75](#).

**Table 75: package\_body Template**

| Attribute                   | Kind     | Limit_Kind                  |
|-----------------------------|----------|-----------------------------|
| subprogram_declaration      | template | subprogram_declaration      |
| subprogram_body             | template | subprogram_body             |
| type_declaration            | template | type_declaration            |
| subtype_declaration         | template | subtype_declaration         |
| constant_declaration        | template | constant_declaration        |
| shared_variable_declaration | template | shared_variable_declaration |
| file_declaration            | template | file_declaration            |
| alias_declaration           | template | alias_declaration           |
| use_clause                  | template | use_clause                  |
| group_template_declaration  | template | group_template_declaration  |

**Table 75: package\_body Template (Continued)**

| Attribute         | Kind     | Limit_Kind        |
|-------------------|----------|-------------------|
| group_declaration | template | group_declaration |
| design_unit       | template | design_unit       |
| header_comment    | template | header_comment    |
| statement_format  | template | statement_format  |
| file_name         | template | ID                |
| ncs_file_name     | template | ID                |
| file_layout       | template | file_layout       |
| line_count        | max/min  | N/A               |
| file_length       | max/min  | N/A               |

Employ the use-clause attribute to limit use-clauses inside the declaration. To constrain use and library clauses appearing before the declaration, employ the design\_unit template instead.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the file\_name attribute to configure the name of containing file, as shown in the following example:

```
limit file_name in package_body to "<package>.Body.vhd"
```

- Use the line\_count attribute to specify the maximum or minimum lines of code in a unit.

## Package Declaration Template

The LRM (§2.5) defines the grammar for package declaration as follows:

```

package_declaration ::=
 package identifier is
 package_declarative_part
 end [package] [package_smple_name] ;

package_declarative_part ::=
 { package_declarative_item }

package_declarative_item ::=
 subprogram_declaration
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | signal_declaration
 | shared_variable_declaration
 | file_declaration
 | alias_declaration
 | component_declaration
 | attribute_declaration
 | attributepecification
 | disconnectionpecification
 | use_clause
 | group_template_declaration
 | group declaration

```

The package\_declaration template is a primary template belonging to the REGION\_PART and OBJECT\_ITEM classes. It contains the attributes shown in [Table 76](#).

**Table 76: package\_declaration Template**

| Attribute                   | Kind     | Limit_Kind                  |
|-----------------------------|----------|-----------------------------|
| cased_identifier            | template | ID                          |
| identifier                  | template | ID                          |
| subprogram_declaration      | template | subprogram_declaration      |
| type_declaration            | template | type_declaration            |
| subtype_declaration         | template | subtype_declaration         |
| constant_declaration        | template | constant_declaration        |
| shared_variable_declaration | template | shared_variable_declaration |

**Table 76: package\_declaration Template (Continued)**

| Attribute                   | Kind      | Limit_Kind                  |
|-----------------------------|-----------|-----------------------------|
| signal_declaration          | template  | signal_declaration          |
| file_declaration            | template  | file_declaration            |
| alias_declaration           | template  | alias_declaration           |
| component_declaration       | template  | component_declaration       |
| attribute_declaration       | template  | attribute_declaration       |
| attribute_specification     | template  | attribute_specification     |
| declaration_profile_s       | aggregate | DECLARATIVE_ITEM            |
| disconnection_specification | template  | disconnection_specification |
| file_length                 | max/min   | N/A                         |
| use_clause                  | template  | use_clause                  |
| group_template_declaration  | template  | group_template_declaration  |
| group_declaration           | template  | group_declaration           |
| header_comment              | template  | header_comment              |
| design_unit                 | template  | design_unit                 |
| statement_format            | template  | statement_format            |
| file_name                   | template  | ID                          |
| ncs_file_name               | template  | ID                          |
| file_layout                 | template  | file_layout                 |
| line_count                  | max/min   | N/A                         |

Employ the use-clause attribute to limit the use-clauses inside the declaration. To constrain use and library clauses appearing before the declaration, use the design\_unit template instead.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the file\_name attribute to configure the name of the containing file, as shown in the following example:

```
limit file_name in package_declaration to "<package>.vhd"
```

- Use the `line_count` attribute to specify the maximum or minimum lines of code in a unit.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is package_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is package_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Physical Type Definition Template

The LRM (§3.1.3) defines the grammar for physical type definition as follows:

```
physical_type_definition ::=
 range_constraint
 units
 primary_unit_declaration
 { secondary_unit_declaration }
 end units [physical_type_simple_name]

primary_unit_declaration ::= identifier

secondary_unit_declaration ::= identifier = physical_literal ;

physical_literal ::= [abstract_literal] unit_name
```

The `physical_type_definition` template is a primary template belonging to no classes. It contains the attributes shown in [Table 77](#).

**Table 77: physical\_type\_definition Template**

| Attribute                     | Kind     | Limit_Kind |
|-------------------------------|----------|------------|
| <code>cased_identifier</code> | template | ID         |
| <code>range</code>            | template | range      |
| <code>identifier</code>       | template | ID         |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is physical_type_definition
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is physical_type_definition
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Port Declaration Template

The `port_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 78](#).

**Table 78: port\_declaration Template**

| Attribute                       | Kind     | Limit_Kind                      |
|---------------------------------|----------|---------------------------------|
| <code>cased_identifier</code>   | template | ID                              |
| <code>identifier</code>         | template | ID                              |
| <code>subtype_indication</code> | template | <code>subtype_indication</code> |
| <code>default</code>            | template | EXPRESSION                      |
| <code>declarative_region</code> | template | REGION_PART                     |
| <code>signal</code>             | local    | N/A                             |
| <code>in</code>                 | local    | N/A                             |
| <code>out</code>                | local    | N/A                             |
| <code>inout</code>              | local    | N/A                             |
| <code>buffer</code>             | local    | N/A                             |
| <code>linkage</code>            | local    | N/A                             |
| <code>bus</code>                | local    | N/A                             |
| <code>is_clock</code>           | local    | ID                              |
| <code>is_reset</code>           | local    | ID                              |
| <code>signals_driven</code>     | template | DRIVEN_OBJECT                   |

**Table 78: port\_declaration Template (Continued)**

| Attribute                | Kind     | Limit_Kind |
|--------------------------|----------|------------|
| driving_expression       | template | EXPRESSION |
| flipflop                 | template | flipflop   |
| latch                    | template | latch      |
| tristate                 | local    | N/A        |
| one_declaration_per_line | local    | N/A        |
| outputs_driven           | max/min  | N/A        |
| register                 | local    | N/A        |
| comment                  | local    | N/A        |
| connections              | max/min  | N/A        |

**Note**

The port\_declaration template contains the same set of attributes as the interface\_signal\_declaration template.

To force the presence of the “in” mode in every port declaration, you could write:

```
force in in port_declaration
```

Because “in” is the default mode, the others must be present.

Use the declarative\_region attribute to limit the regions where this item can be declared. You can use the declarative\_regions attribute to point to a list of templates of the REGION\_PART class.

Use the register and bus attributes to control the presence of the register and bus VHDL keywords.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the signals\_driven attribute to control the signals driven by the current signal.
- Use the driving\_expression attribute to control the expressions accepted as drivers for the current signal.
- Use the flipflop attribute to control whether or not the signal infers a flipflop.
- Use the latch attribute to control whether or not the signal infers a latch.



- Use the `tristate` attribute to constrain tristate signals.
- Use the `connections` attribute to control the number of times the signal drives or is driven by other signals.
- Use the `outputs_driven` attribute to constrain the number of directly driven outputs. For example, in the following code, the number of outputs driven by `d1` is 1, whereas the number driven by `d2` is 0.

```

q1<=d1; --d1 drives port q1 directly
if cond then --infers mux
 q2<=d2; --d2 is input to mux driving port q2
else
 q2<='1';
end if;

```

To write a rule forbidding the use of feedthrough ports, you could use the following template:

```

template FEEDTHRU_PORT is port_declaration
 max outputs_driven is 0
end

```

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```

template T1 is interface_signal_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is interface_signal_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The identifier attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Procedure Call Statement Template

The LRM (§8.6) defines the grammar for procedure call statement as follows:

```
procedure_call_statement ::= [label :] procedure_call ;
procedure_call ::= procedure_name [(actual_parameter_part)]
```

The `procedure_call_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 79](#).

**Table 79: procedure\_call\_statement Template**

| Attribute             | Kind     | Limit_Kind       |
|-----------------------|----------|------------------|
| label                 | template | ID               |
| cased_identifier      | template | ID               |
| identifier            | template | ID               |
| actual_parameter_part | template | association_list |
| declarative_region    | template | REGION_PART      |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is procedure_call_statement
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is procedure_call_statement
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Process Statement Template

The LRM (§9.2) defines the grammar for process statement as follows:

```

process_statement ::=
 [process_label :]
 [postponed] process [(sensitivity_list)] [is]
 process_declarative_part
 begin
 process_statement_part
 end [postponed] process [process_label] ;
process_declarative_part ::=
 { process_declarative_item }
process_declarative_item ::=
 subprogram_declaration
 | subprogram_body
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | variable_declaration
 | file_declaration
 | alias_declaration
 | attribute_declaration
 | attribute_specification
 | use_clause
 | group_type_declaration
 | group_declaration
process_statement_part ::=
 { sequential_statement }

```

The process\_statement template is a primary template belonging to the CONCURRENT\_STATEMENT class. It contains the attributes shown in [Table 80](#).

**Table 80: process\_statement Template**

| Attribute              | Kind     | Limit_Kind             |
|------------------------|----------|------------------------|
| cased_label            | template | ID                     |
| label                  | template | ID                     |
| sensitivity            | template | EXPRESSION             |
| subprogram_declaration | template | subprogram_declaration |
| subprogram_body        | template | subprogram_body        |
| type_declaration       | template | type_declaration       |
| subtype_declaration    | template | subtype_declaration    |
| constant_declaration   | template | constant_declaration   |

**Table 80: process\_statement Template (Continued)**

| Attribute                     | Kind     | Limit_Kind                    |
|-------------------------------|----------|-------------------------------|
| variable_declaration          | template | variable_declaration          |
| file_declaration              | template | file_declaration              |
| alias_declaration             | template | alias_declaration             |
| attribute_declaration         | template | attribute_declaration         |
| attribute_specification       | template | attribute_specification       |
| use_clause                    | template | use_clause                    |
| group_template_declaration    | template | group_template_declaration    |
| group_declaration             | template | group_declaration             |
| unused_declaration            | local    | N/A                           |
| assertion_statement           | template | assertion_statement           |
| report_statement              | template | report_statement              |
| procedure_call_statement      | template | procedure_call_statement      |
| if_statement                  | template | if_statement                  |
| signal_assignment_statement   | template | signal_assignment_statement   |
| variable_assignment_statement | template | variable_assignment_statement |
| case_statement                | template | case_statement                |
| loop_statement                | template | loop_statement                |
| for_loop_statement            | template | for_loop_statement            |
| while_loop_statement          | template | while_loop_statement          |
| next_statement                | template | next_statement                |
| exit_statement                | template | exit_statement                |
| return_statement              | template | return_statement              |
| wait_statement                | template | wait_statement                |
| null_statement                | local    | N/A                           |
| postponed                     | local    | N/A                           |

**Table 80: process\_statement Template (Continued)**

| Attribute                           | Kind      | Limit_Kind           |
|-------------------------------------|-----------|----------------------|
| header_comment                      | template  | header_comment       |
| latch                               | template  | latch                |
| flipflop                            | template  | flipflop             |
| fsm                                 | local     | N/A                  |
| synchronous_reset                   | template  | synchronous_reset    |
| asynchronous_reset                  | template  | asynchronous_reset   |
| statement_format                    | template  | statement_format     |
| declarative_region                  | template  | REGION_PART          |
| combinatorial                       | local     | N/A                  |
| complete_sensitivity                | local     | N/A                  |
| missing_signals_in_sensitivity_list | local     | N/A                  |
| redundancy_in_sensitivity_list      | local     | N/A                  |
| initialize_signals                  | local     | N/A                  |
| fully_assign_signals                | local     | N/A                  |
| fully_assign_variables              | local     | N/A                  |
| initialize_variables                | local     | N/A                  |
| maximum_variable_usage              | local     | N/A                  |
| dead_code                           | local     | N/A                  |
| sensitivity_count                   | max/min   | N/A                  |
| clock_expression_count              | max/min   | N/A                  |
| clock_signal                        | max/min   | N/A                  |
| statement_profile_s                 | aggregate | SEQUENTIAL_STATEMENT |

Use the `statement_profile_s` attribute to dictate the order of statements. For example, to force synchronous processes to appear before asynchronous processes, you could write the following rule:

```
limit statement_profile_s in process_statement to
 first {VAR_ASS_TPL} then IF_STM, others {}
 severity error
```

where `VAR_ASS_TPL` and `IF_STM` are two valid sequential statement templates. The `{}` after the `others` keyword means that any statement is accepted. You can only use this notation for the last set of templates.

Use the `latch` and `flipflop` attributes to control the different types of memory inferred. For example, to forbid the use of latches in a process, you could write the following rule:

```
no latch in process_statement
```

Use the `fsm` attribute to test if the process statement belongs to a finite state machine (FSM). For example, the following VerSL flags process statements that are not part of an FSM:

```
force fsm in process_statement
```

Use the `combinatorial` attribute to force or forbid the presence of memory elements in a process. Note that a combinatorial process is defined as one that has no clock.

Use the `clock_expression_count` attribute to control the number of different clock expressions in a subprogram (procedure) body.

Use `clock_signal` attribute to control the number of different clock signals in a subprogram (procedure) body.

If you force the `complete_sensitivity` attribute to be present, Leda ensures that every object in the sensitivity list is read in the process and that every object read is in the sensitivity list, including objects that are also modified.

The `initialize_variables` and `initialize_signals` attributes cause Leda to generate an error if, in a combinatorial process, a variable or signal is read before being written on any flow of control, or if it is read without being written.

If you force the `fully_assign_signals` attribute, Leda ensures that every signal assigned in the process is fully assigned on every possible flow of control through the process.

If you force the `maximum_variable_usage` attribute, Leda ensures that wherever possible, variables are used instead of signals.

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is process_statement
 limit label to "^[a-z] [a-z0-9_]*$"
end
template T2 is process_statement
 limit cased_label to "^[a-z] [a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Range Template

The LRM (§3.1) defines the grammar for range constraint as follows:

```
range_constraint ::= range range
range ::= range_attribute_name | simple_expression direction
 simple_expression
direction ::= to | downto
```

The range template is a primary template belonging to the `DISCRETE_RANGE` class. It contains the attributes shown in [Table 81](#).

**Table 81: range Template**

| Attribute                     | Kind     | Limit_Kind |
|-------------------------------|----------|------------|
| <code>left_expression</code>  | template | EXPRESSION |
| <code>right_expression</code> | template | EXPRESSION |
| <code>range_attribute</code>  | local    | N/A        |
| <code>range_index</code>      | local    | N/A        |
| <code>null_range</code>       | local    | N/A        |
| <code>to</code>               | local    | N/A        |
| <code>downto</code>           | local    | N/A        |
| <code>evaluation_time</code>  | set      | N/A        |
| <code>high_bound</code>       | max/min  | N/A        |
| <code>low_bound</code>        | max/min  | N/A        |

For example, the following rule:

```
no range_attribute in range severity error
```

means that any time a `range_attribute` appears in the VHDL code, Leda flags an error.

Whereas, this next rule:

```
no null_range in range severity error
```

means that ranges of the form 3 to 0 or 0 downto 5 cause Leda to flag an error.

The following rule:

```
no downto in range severity error
```

means that the presence of the `downto` VHDL keyword causes Leda to flag an error. The same is true for the `to` attribute and the presence of the `to` VHDL keyword in a range expression.

For the following rule:

```
set evaluation_time in range to locally_static_evaluation
 severity error
```

both left and right range expressions must be locally static; otherwise, Leda flags an error.

Use the `high_bound` and `low_bound` attributes to specify fixed, static ranges without defining literal templates. For example, the following rules:

```
max low_bound in range is 0 severity error
no downto in range severity error
```

mean that all ranges must be ascending and start at 0.

To build more complex limitations for either range expression, use the `left_expression` and `right_expression` attributes for the left-hand and right-hand expressions, respectively. To limit these attributes, point them to templates of type expression.



## Record Type Definition Template

The LRM (§3.2.2) defines the grammar for record type definition as follows:

```
record_type_definition ::=
 record
 element_declaration
 { element_declaration }
 end record [record_type_simple_name]

element_declaration ::=
 identifier_list : element_subtype_definition ;
identifier_list ::= identifier { , identifier }
element_subtype_definition ::= subtype_indication
```

The record\_type\_definition template is a primary template belonging to no classes. It contains the attributes shown in [Table 82](#).

**Table 82: record\_type\_definition Template**

| Attribute           | Kind     | Limit_Kind          |
|---------------------|----------|---------------------|
| element_declaration | template | element_declaration |
| type_mark           | template | TYPE_MARK           |

## Report Statement Template

The LRM (§8.3) defines the grammar for report statement as follows:

```
report_statement ::= [label :] report expression [severity expression]
```

The report\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 83](#).

**Table 83: report\_statement Template**

| Attribute           | Kind     | Limit_Kind  |
|---------------------|----------|-------------|
| cased_label         | template | ID          |
| label               | template | ID          |
| report_expression   | template | EXPRESSION  |
| severity_expression | template | EXPRESSION  |
| declarative_region  | template | REGION_PART |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is report_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is report_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Return Statement Template

The LRM (§8.12) defines the grammar for return statement as follows:

```
return_statement ::= [label :] return expression ;
```

The `return_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 84](#).

**Table 84: return\_statement Template**

| Attribute                       | Kind     | Limit Kind  |
|---------------------------------|----------|-------------|
| <code>cased_label</code>        | template | ID          |
| <code>label</code>              | template | ID          |
| <code>expression</code>         | template | EXPRESSION  |
| <code>declarative_region</code> | template | REGION_PART |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is return_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is return_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Selected Name Template

The LRM (§6.3) defines the grammar for selected name as follows:

```

selected_name ::= prefix . suffix
suffix ::=
 simple_name
 | character_literal
 | operator_symbol
 | all

```

The `selected_name` template is a secondary template belonging to the `EXPRESSION`, `NAME`, and `TARGET` classes. It contains the attributes shown in [Table 85](#).

**Table 85: selected\_name Template**

| Attribute                      | Kind     | Limit_Kind  |
|--------------------------------|----------|-------------|
| <code>cased_identifier</code>  | template | ID          |
| <code>identifier</code>        | template | ID          |
| <code>object_definition</code> | template | OBJECT_ITEM |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the `identifier` is case-sensitive. Consider the following templates:

```

template T1 is selected_name
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is selected_name
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier will match both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Selected Signal Assignment Template

The LRM (§9.5.2) defines the grammar for selected signal assignment as follows:

```
selected_signal_assignment ::=
 with expression select
 target <= options selected_waveforms ;
```

The `selected_signal_assignment` template is a primary template belonging to the `CONCURRENT_STATEMENT` class. It contains the attributes shown in [Table 86](#).

**Table 86: selected\_signal\_assignment Template**

| Attribute                          | Kind      | Limit_Kind                      |
|------------------------------------|-----------|---------------------------------|
| <code>cased_label</code>           | template  | ID                              |
| <code>label</code>                 | template  | ID                              |
| <code>target</code>                | template  | TARGET                          |
| <code>reject_expression</code>     | template  | EXPRESSION                      |
| <code>expression</code>            | template  | EXPRESSION                      |
| <code>operand_size_mismatch</code> | local     | N/A                             |
| <code>postponed</code>             | local     | N/A                             |
| <code>range_overflow</code>        | local     | N/A                             |
| <code>statement_format</code>      | template  | <code>statement_format</code>   |
| <code>transport</code>             | local     | N/A                             |
| <code>inertial</code>              | local     | N/A                             |
| <code>guarded</code>               | local     | N/A                             |
| <code>read_write</code>            | local     | N/A                             |
| <code>waveform_count</code>        | max/min   | N/A                             |
| <code>selected_waveforms_s</code>  | aggregate | <code>selected_waveforms</code> |

Use the `transport`, `inertial`, and `guarded` attributes to force or forbid the presence of the corresponding VHDL keywords (`transport`, `inertial`, and `guarded`).

Use the `waveform_count` attribute to set the maximum number of waveforms allowed.

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is selected_signal_assignment
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is selected_signal_assignment
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Selected Waveforms Template

The LRM (§9.5.2) defines the grammar for selected waveforms as follows:

```
selected_waveforms ::=
 { waveform when choices , }
 waveform when choices
```

The `selected_waveforms` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 87](#) and is used in the [Selected Signal Assignment Template](#).

**Table 87: selected\_waveforms Template**

| Attribute | Kind     | Limit_Kind |
|-----------|----------|------------|
| waveform  | template | waveform   |
| choice    | template | EXPRESSION |
| others    | local    | N/A        |

Use the `others` attribute to force or forbid the presence of the `others` VHDL keyword.

## Shared Variable Declaration Template

The `shared_variable_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 88](#).

**Table 88: `shared_variable_declaration` Template**

| Attribute                             | Kind     | Limit_Kind                      |
|---------------------------------------|----------|---------------------------------|
| <code>cased_identifier</code>         | template | ID                              |
| <code>identifier</code>               | template | ID                              |
| <code>subtype_indication</code>       | template | <code>subtype_indication</code> |
| <code>default</code>                  | template | EXPRESSION                      |
| <code>declarative_region</code>       | template | REGION_PART                     |
| <code>one_declaration_per_line</code> | local    | N/A                             |
| <code>comment</code>                  | local    | N/A                             |

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the the identifier is case-sensitive. Consider the following templates:

```

template T1 is variable_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is variable_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Signal Assignment Statement Template

The LRM (§8.4) defines the grammar for signal assignment statement as follows:

```

signal_assignment_statement ::=
 [label :] target <= [delay_mechanism] waveform ;
target ::=
 name
 | aggregate
delay_mechanism ::=
 transport
 | [reject time_expression] inertial

```

The `signal_assignment_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 89](#).

**Table 89: `signal_assignment_statement` Template**

| Attribute                          | Kind     | Limit_Kind  |
|------------------------------------|----------|-------------|
| <code>cased_label</code>           | template | ID          |
| <code>label</code>                 | template | ID          |
| <code>target</code>                | template | TARGET      |
| <code>waveform</code>              | template | waveform    |
| <code>operand_size_mismatch</code> | local    | N/A         |
| <code>reject_expression</code>     | template | EXPRESSION  |
| <code>declarative_region</code>    | template | REGION_PART |
| <code>range_overflow</code>        | local    | N/A         |
| <code>transport</code>             | local    | N/A         |
| <code>inertial</code>              | local    | N/A         |
| <code>read_write</code>            | local    | N/A         |
| <code>waveform_count</code>        | max/min  | N/A         |

Use the `transport` and `inertial` keywords to force or forbid the presence of the corresponding `transport` and `inertial` VHDL keywords.

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is signal_assignment_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is signal_assignment_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Signal Declaration Template

The LRM (§4.3.1.2) defines the grammar for signal declaration as follows:

```
signal_declaration ::=
 signal identifier_list:subtype_indication [signal_kind]
 [:= expression] ;
signal_kind ::= register | bus
```

The `signal_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 90](#).

**Table 90: signal\_declaration Template**

| Attribute                       | Kind     | Limit_Kind         |
|---------------------------------|----------|--------------------|
| <code>cased_identifier</code>   | template | ID                 |
| <code>identifier</code>         | template | ID                 |
| <code>subtype_indication</code> | template | subtype_indication |
| <code>default</code>            | template | EXPRESSION         |
| <code>declarative_region</code> | template | REGION_PART        |
| <code>outputs_driven</code>     | max/min  | N/A                |
| <code>register</code>           | local    | N/A                |
| <code>bus</code>                | local    | N/A                |
| <code>signals_driven</code>     | template | DRIVEN_OBJECT      |
| <code>driving_expression</code> | template | EXPRESSION         |
| <code>flipflop</code>           | template | flipflop           |
| <code>latch</code>              | template | latch              |



**Table 90: signal\_declaration Template (Continued)**

| Attribute                | Kind  | Limit_Kind |
|--------------------------|-------|------------|
| one_declaration_per_line | local | N/A        |
| comment                  | local | N/A        |
| tristate                 | local | N/A        |
| is_clock                 | local | ID         |
| is_reset                 | local | ID         |

Use the `declarative_region` attribute to limit where this item is allowed. You can point the `declarative_region` attribute at templates of the `REGION_PART` class.

Use the `register` and `bus` attributes to control the presence of the corresponding register and bus VHDL keywords.

There are some application-specific attributes that you can use with this template to make it easier to write commonly-used rules. For example:

- Use the `signals_driven` attribute to control the signals driven by the current signal.
- Use the `driving_expression` attribute to control the expressions allowed as drivers for the current signal.
- Use the `flipflop` attribute to control whether or not the signal infers a flipflop.
- Use the `latch` attribute to control whether or not the signal infers a latch.
- Use the `tristate` attribute to control whether or not the signal infers a latch.
- Use the `outputs_driven` attribute to control the number of directly driven outputs. For example, in the following code the number of outputs driven by `d1` is 1, whereas the number driven by `d2` is 0.

```

q1<=d1; --d1 drives port q1 directly
if cond then --infers mux
 q2<=d2; --d2 is input to mux driving port q2
else
 q2<='1';
end if;

```

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is signal_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is signal_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Simple Name Template

The `simple_name` template is a primary template belonging to the `EXPRESSION`, `NAME`, and `TARGET` classes. It contains the attributes shown in [Table 91](#).

**Table 91: simple\_name Template**

| Attribute                      | Kind     | Limit_Kind  |
|--------------------------------|----------|-------------|
| <code>cased_identifier</code>  | template | ID          |
| <code>identifier</code>        | template | ID          |
| <code>object_definition</code> | template | OBJECT_ITEM |
| <code>flipflop</code>          | template | flipflop    |
| <code>latch</code>             | template | latch       |

Use the `object_definition` attribute to control the type of definition associated with the `simple_name`. For example, to focus just on ports in a given expression, you could write the following template:

```
template CLK_NAME is simple_name
 limit object_definition to port_declaration
end
```

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the `flipflop` attribute to control whether or not the object infers a flipflop.
- Use the `latch` attribute to control whether or not the object infers a latch.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is simple_name
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is simple_name
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Slice Name Template

The LRM (§6.5) defines the grammar for slice name as follows:

```
slice_name ::= prefix (discrete_range)
```

The `slice_name` template is a secondary template belonging to the `EXPRESSION`, `NAME`, and `TARGET` classes. It contains the attributes shown in [Table 92](#).

**Table 92: slice\_name Template**

| Attribute                      | Kind     | Limit_Kind  |
|--------------------------------|----------|-------------|
| <code>cased_identifier</code>  | template | ID          |
| <code>identifier</code>        | template | ID          |
| <code>object_definition</code> | template | OBJECT_ITEM |
| <code>range</code>             | template | range       |
| <code>range_overflow</code>    | local    | N/A         |
| <code>flipflop</code>          | template | flipflop    |
| <code>latch</code>             | template | latch       |

Use the `object_definition` attribute to control the type of definition associated with the `simple_name`. For example, to focus just on ports in a given expression, you could write the following template:

```
template CLK_NAME is simple_name
 limit object_definition to port_declaration
end
```

There are some application-specific associated with this template that make it easier to write commonly-used rules. For example:

- Use the `flipflop` attribute to control whether or not the object infers a flipflop.
- Use the `latch` attribute to control whether or not the object infers a latch.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the `identifier` is case-sensitive. Consider the following templates:

```
template T1 is slice_name
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is slice_name
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Statement Format Template

Use the `statement_format` template to control issues relating to compound statements; that is, statements that contain other statements.

The `statement_format` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 93](#).

**Table 93: `statement_format` Template**

| Attribute                           | Kind    | Limit_Kind |
|-------------------------------------|---------|------------|
| <code>end_label</code>              | local   | N/A        |
| <code>one_statement_per_line</code> | local   | N/A        |
| <code>indentation_depth</code>      | max/min | N/A        |
| <code>line_count</code>             | max/min | N/A        |

Use the `end_label` attribute to force the presence of an ending label. Consider the following rule:

```
template FEL is statement_format
 force end_label
end
limit statement_format in subprogram_body to FEL
```

With this rule activated, the VHDL code must contain the optional `subprogram_kind` clause at the end of the subprogram statement. This means that:

```
procedure P is
begin
end procedure P;
```

is acceptable, whereas:

```
procedure P is
begin
end;
```

generates an error.

Use the `indentation_depth` attribute to control the maximum or minimum number of indentation levels within a compound statement.

Use the `line_count` attribute to control the maximum or minimum number of lines of code within a compound statement.

Use the `one_statement_per_line` attribute to force a single line for each declaration.

## Subprogram Body Template

The LRM (§2.2) defines the grammar for subprogram body as follows:

```
subprogram_body ::=
 subprogram_specification is
 subprogram_declarative_part
 begin
 subprogram_statement_part
 end [subprogram_kind] [designator] ;
```

```
subprogram_declarative_part ::=
 { subprogram_declarative_item }
```

```
subprogram_declarative_item ::=
 subprogram_declaration
 | subprogram_body
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | variable_declaration
 | file_declaration
 | alias_declaration
 | attribute_declaration
 | attribute_specification
 | use_clause
 | group_template_declaration
```

```

| group declaration

subprogram_statement_part ::=
 { sequential_statement }
 { subprogram_sequential_statement }

subprogram_kind ::= procedure | function

```

The `subprogram_body` template is a primary template belonging to the `REGION_PART` class. It contains the attributes shown in [Table 94](#).

**Table 94: subprogram\_body Template**

| Attribute                     | Kind     | Limit_Kind                    |
|-------------------------------|----------|-------------------------------|
| subprogram_declaration        | template | subprogram_declaration        |
| subprogram_body               | template | subprogram_body               |
| type_declaration              | template | type_declaration              |
| subtype_declaration           | template | subtype_declaration           |
| constant_declaration          | template | constant_declaration          |
| variable_declaration          | template | variable_declaration          |
| file_declaration              | template | file_declaration              |
| fully_assign_variables        | local    | N/A                           |
| alias_declaration             | template | alias_declaration             |
| attribute_declaration         | template | attribute_declaration         |
| attribute_specification       | template | attribute_specification       |
| use_clause                    | template | use_clause                    |
| group_template_declaration    | template | group_template_declaration    |
| group_declaration             | template | group_declaration             |
| assertion_statement           | template | assertion_statement           |
| report_statement              | template | report_statement              |
| procedure_call_statement      | template | procedure_call_statement      |
| if_statement                  | template | if_statement                  |
| signal_assignment_statement   | template | signal_assignment_statement   |
| variable_assignment_statement | template | variable_assignment_statement |

**Table 94: subprogram\_body Template (Continued)**

| Attribute                 | Kind      | Limit_Kind           |
|---------------------------|-----------|----------------------|
| case_statement            | template  | case_statement       |
| loop_statement            | template  | loop_statement       |
| for_loop_statement        | template  | for_loop_statement   |
| while_loop_statement      | template  | while_loop_statement |
| next_statement            | template  | next_statement       |
| exit_statement            | template  | exit_statement       |
| return_statement          | template  | return_statement     |
| wait_statement            | template  | wait_statement       |
| null_statement            | local     | N/A                  |
| header_comment            | template  | header_comment       |
| statement_format          | template  | statement_format     |
| out_params_fully_assigned | local     | N/A                  |
| dead_code                 | local     | N/A                  |
| return_last               | local     | N/A                  |
| recursion_type            | set       | N/A                  |
| clock_expression_count    | max/min   | N/A                  |
| clock_signal              | max/min   | N/A                  |
| statement_profile_s       | aggregate | N/A                  |
| return_fully_assigned     | local     | N/A                  |

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the header\_comment attribute to control keywords in comments at the start of units or subprograms.
- Use the statement\_format attribute to handle organization issues within compound statements.
- Use the clock\_expression\_count attribute to control the number of different clock expressions allowed in a subprogram (procedure) body.

- Use the `clock_signal` attribute to control the number of different clock signals allowed in a subprogram (procedure) body.
- Use the `out_params_fully_assigned` attribute to ensure that every parameter of mode “out” is assigned on every flow of execution within the subprogram. For example:

```
force out_params_fully_assigned in subprogram_body
```

- Use the `return_last` attribute to force the last statement of a function to be a return statement. For example, if you want functions to contain only one return statement as the last statement in the function, you could write the following template for a rule:

```
template ONE-RETURN is subprogram_body
 force return_last
 max return_statement is 1
end
```

- Use the `recursion_type` attribute to specify different types of recursion that are allowed in subprograms. This attribute points to an element of the primitive type `rsp_Recursion_Type`. For example:

```
set recursion_type in subprogram_body to static_recursion
severity ERROR
```

This rule forces the Leda Checker to test each subprogram. If a recursive call is found enclosed within a loop having non-static bounds, Leda flags an error.

- Use the `fully_assign_variables` attribute to ensure that all variables (inside a subprogram or a sequential block) are assigned on every possible flow of control through the sequential block. For example:

```
force fully_assign_variables in subprogram_body
```

```
--Fail case
function test --FAIL
(D_IN : in std_logic;
 RST : in std_logic)
return std_logic is variable O : std_logic;
begin
 if (RST = '1') then
 O := D_IN;
 end if;
 return O;
end test;
```

```
force fully_assign_variables in subprogram_body
message "Assignment to variables missing inside subprogram"
severity WARNING
```



- Use the `out_params_fully_assigned` attribute to ensure that all subprogram parameters of mode out or inout (VHDL) are assigned on every possible flow of control through the subprogram. For example:

```
force out_params_fully_assigned in subprogram_body

--Fail case
procedure test --FAIL
(D_IN : in std_logic;
 RST : in std_logic;
 D_OUT : out std_logic) is
begin
 if (RST = '1') then
 D_OUT := D_IN;
 end if;
end test;

force out_params_fully_assigned in subprogram_body
message "Assignment to out or inout parameters missing inside
subprogram"
severity WARNING
```

- Use the `return_fully_assigned` attribute to ensure that there is a return statement on every possible flow of control through the function. For example:

```
force return_fully_assigned in subprogram_body

function dlatch --FAIL
(D : in std_logic_vector(3 downto 0);
 G : in std_logic;
 RST : in std_logic)
return std_logic_vector is
variable ret : std_logic_vector(3 downto 0);
begin
 if (RST = '1') then
 return "0000";
 elsif (G = '1') then
 return D;
 end if;
end dlatch;

force return_fully_assigned in subprogram_body
message "No value returned for a particular flow through the
function"
severity WARNING
```

## Subprogram Declaration Template

The LRM (§2.1) defines the grammar for subprogram declaration as follows:

```

subprogram_declaration ::=
 subprogram_specification ;
subprogram_specification ::=
 procedure_designator [(formal_parameter_list)]
 | [pure | impure] function_designator [(
formal_parameter_list)]
 return_type_mark
designator ::= identifier | operator_symbol
formal_parameter_list ::= parameter_interface_list

```

The `block_specification` template is a primary template belonging to the `REGION_PART` and `OBJECT_ITEM` classes. It contains the attributes shown in [Table 95](#).

**Table 95: subprogram\_declaration Template**

| Attribute                       | Kind     | Limit_Kind                    |
|---------------------------------|----------|-------------------------------|
| <code>cased_identifier</code>   | template | ID                            |
| <code>identifier</code>         | template | ID                            |
| <code>formal_parameter</code>   | template | <code>formal_parameter</code> |
| <code>type_mark</code>          | template | TYPE_MARK                     |
| <code>subprogram_body</code>    | template | <code>subprogram_body</code>  |
| <code>declarative_region</code> | template | REGION_PART                   |
| <code>operator_symbol</code>    | local    | N/A                           |
| <code>procedure</code>          | local    | N/A                           |
| <code>function</code>           | local    | N/A                           |
| <code>impure</code>             | local    | N/A                           |
| <code>pure</code>               | local    | N/A                           |
| <code>header_comment</code>     | template | ID                            |
| <code>parameter_count</code>    | max/min  | N/A                           |

Use the `declarative_region` attribute to point to a template from the class `REGION_PART` that constrains the enclosing region of the current template. For example:

```
limit declarative_part in subprogram_declaration to
 package_declaration severity error
```

means that subprogram declarations can only appear in package declaration regions.

Use the `operator_symbol` attribute to force or forbid the use of operator symbols as subprogram names. For example:

```
no operator_symbol in subprogram_declaration severity error
```

Use the `procedure` and `function` attributes to force or forbid the presence of procedure or function subprograms. For example, the following command:

```
no procedure in subprogram_declaration severity error
```

means that procedure declarations are not allowed.

There are some application-specific attributes associated with this template that make it easier to write commonly-used rules. For example:

- Use the `parameter_count` attribute to set the maximum or minimum number of parameters allowed.
- Use the `header_comment` attribute to control the keywords in comments at the start of units or subprograms.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is subprogram_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is subprogram_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Subtype Declaration Template

The LRM (§4.2) defines the grammar for subtype declaration as follows:

```
subtype_declaration ::= subtype identifier is subtype_indication ;
```

The subtype\_declaration template is a primary template belonging to the TYPE\_MARK class. It contains the attributes shown in [Table 96](#).

**Table 96: subtype\_declaration Template**

| Attribute          | Kind     | Limit_Kind         |
|--------------------|----------|--------------------|
| cased_identifier   | template | ID                 |
| identifier         | template | ID                 |
| subtype_indication | template | subtype_indication |
| declarative_region | template | REGION_PART        |

Use the cased\_identifier attribute just like the identifier attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is subtype_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is subtype_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The id1 identifier matches both templates, whereas the ID1 identifier only matches the first template. The identifier attribute is not case-sensitive because VHDL is not case-sensitive.

## Subtype Indication Template

The LRM (§4.2) defines the grammar for subtype indication as follows:

```

subtype_indication ::=
 [resolution_function_name] type_mark [constraint]
type_mark ::=
 type_name
 | subtype_name
constraint ::=
 range_constraint
 | index_constraint

```

The subtype\_indication template is a primary template belonging to the DISCRETE\_RANGE class. It contains the attributes shown in [Table 97](#).

**Table 97: subtype\_indication Template**

| Attribute           | Kind     | Limit_Kind         |
|---------------------|----------|--------------------|
| type_mark           | template | TYPE_MARK          |
| subtype_indication  | template | subtype_indication |
| range               | template | range              |
| resolution_function | local    | N/A                |
| evaluation_time     | set      | N/A                |

Use the subtype\_indication and range attributes to constrain the subtype.

Use the resolution\_function attribute to force or forbid the presence of resolution function names in subtype indications.

## Synchronous Initialization Template

The `synchronous_initialization` template is a primary template belonging to no classes. It contains the attributes shown in [Table 98](#).

**Table 98: synchronous\_initialization Template**

| Attribute                      | Kind     | Limit_Kind                     |
|--------------------------------|----------|--------------------------------|
| <code>cased_identifier</code>  | template | ID                             |
| <code>identifier</code>        | template | ID                             |
| <code>is_load</code>           | local    | N/A                            |
| <code>is_reset</code>          | local    | N/A                            |
| <code>is_set</code>            | local    | N/A                            |
| <code>edge</code>              | set      | N/A                            |
| <code>expression</code>        | template | EXPRESSION                     |
| <code>object_definition</code> | template | OBJECT_ITEM                    |
| <code>connectivity_path</code> | template | <code>connectivity_path</code> |
| <code>gated_in_unit</code>     | template | ID                             |

Leda recognizes all RTL reset expressions. Use the `expression` attribute to limit the expression type.

Use the `object_definition` attribute to limit the type of the reset signal.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```

template T1 is synchronous_initialization
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is synchronous_initialization
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end

```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

**Note**

When Leda encounters a complex expression (containing more than one object) used as a clock or reset, naming convention rules are automatically deactivated. This is because Leda assumes that the clock or reset is the output of a logic gate inferred by the complex expression and not simply one of the objects in the expression.

## Type Declaration Template

The LRM (§4.1) defines the grammar for type declaration as follows:

```

type_declaration ::=
 full_type_declaration
 | incomplete_type_declaration
full_type_declaration ::=
 type_identifier is type_definition ;
type_definition ::=
 scalar_type_definition
 | composite_type_definition
 | access_type_definition
 | file_type_definition

```

The `type_declaration` template is a primary template belonging to the `TYPE_MARK` class. It contains the attributes shown in [Table 99](#).

**Table 99: `type_declaration` Template**

| Attribute                                | Kind     | Limit_Kind      |
|------------------------------------------|----------|-----------------|
| <code>cased_identifier</code>            | template | ID              |
| <code>identifier</code>                  | template | ID              |
| <code>type_definition</code>             | template | TYPE_DEFINITION |
| <code>declarative_region</code>          | template | REGION_PART     |
| <code>incomplete_type_declaration</code> | local    | N/A             |

Use the `incomplete_type_declaration` attribute to force or forbid the presence of incomplete type declarations.

Use the `cased_identifier` attribute just like the `identifier` attribute, but in cases where the identifier is case-sensitive. Consider the following templates:

```
template T1 is type_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is type_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The `identifier` attribute is not case-sensitive because VHDL is not case-sensitive.

## Unconstrained Array Definition Template

The LRM (§3.2.1) defines the grammar for unconstrained array definition as follows:

```
unconstrained_array_definition ::=
 array (index_subtype_definition { , index_subtype_definition })
 of element_subtype_indication
```

The `unconstrained_array_definition` template is a primary template belonging to no classes. It contains the attributes shown in [Table 100](#).

**Table 100: unconstrained\_array\_definition Template**

| Attribute                         | Kind      | Limit_Kind                      |
|-----------------------------------|-----------|---------------------------------|
| <code>subtype_indication</code>   | template  | <code>subtype_indication</code> |
| <code>dimension_count</code>      | max/min   | N/A                             |
| <code>subtype_definition_s</code> | aggregate | <code>subtype_indication</code> |

Use the `dimension_count` attribute to set the maximum or minimum number of dimensions allowed. For example, the command:

```
max dimension_count in constrained_array_definition is 1
```

means that only 1-dimensional arrays are allowed.

To allow more than one dimension, put different constraints on the subtype definitions for each dimension. Each aggregate must point to a set of `index_subtype_definition` templates.



## Use Clause Template

The LRM (§10.4) defines the grammar for use clause as follows:

```
use_clause ::= use selected_name { , selected_name } ;
```

The use\_clause template is a primary template belonging to no classes. It contains the attribute shown in [Table 101](#).

**Table 101: use\_clause Template**

| Attribute     | Kind     | Limit_Kind    |
|---------------|----------|---------------|
| selected_name | template | selected_name |

## Variable Assignment Statement Template

The LRM (§8.5) defines the grammar for variable assignment statement as follows:

```
variable_assignment_statement ::= [label :] target := expression ;
```

The variable\_assignment\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 102](#).

**Table 102: variable\_assignment\_statement Template**

| Attribute             | Kind     | Limit_Kind  |
|-----------------------|----------|-------------|
| cased_label           | template | ID          |
| label                 | template | ID          |
| operand_size_mismatch | local    | N/A         |
| range_overflow        | local    | N/A         |
| target                | template | TARGET      |
| expression            | template | EXPRESSION  |
| declarative_region    | template | REGION_PART |
| read_write            | local    | N/A         |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is variable_assignment_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is variable_assignment_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Variable Declaration Template

The LRM (§4.3.1.3) defines the grammar for variable declaration as follows:

```
variable_declaration ::=
 [shared] variable identifier_list:subtype_indication [:= expression]
```

The `variable_declaration` template is a primary template belonging to the `OBJECT_ITEM` class. It contains the attributes shown in [Table 103](#).

**Table 103: variable\_declaration Template**

| Attribute                             | Kind     | Limit_Kind                      |
|---------------------------------------|----------|---------------------------------|
| <code>cased_identifier</code>         | template | ID                              |
| <code>identifier</code>               | template | ID                              |
| <code>subtype_indication</code>       | template | <code>subtype_indication</code> |
| <code>default</code>                  | template | EXPRESSION                      |
| <code>declarative_region</code>       | template | REGION_PART                     |
| <code>one_declaration_per_line</code> | local    | N/A                             |
| <code>comment</code>                  | local    | N/A                             |

Use `cased_identifier` attribute just like the `identifier` attribute, but in cases where the `identifier` is case-sensitive. Consider the following templates:

```
template T1 is variable_declaration
 limit identifier to "^[a-z][a-z0-9_]*$"
end
template T2 is variable_declaration
 limit cased_identifier to "^[a-z][a-z0-9_]*$"
end
```

The `id1` identifier matches both templates, whereas the `ID1` identifier only matches the first template. The identifier attribute is not case-sensitive because VHDL is not case-sensitive.

Use the `one_declaration_per_line` attribute to force a single line for each declaration.

Use the `comment` attribute to force a comment on the same line as the declaration.

## Wait Statement Template

The LRM (§8.1) defines the grammar for wait statement as follows:

```
wait_statement ::=
 [label :] wait [sensitivity_clause] [condition_clause]
 [timeout_clause] ;
sensitivity_clause ::= on sensitivity_list
sensitivity_list ::= signal_name { , signal_name }
condition_clause ::= until condition
condition ::= boolean_expression
timeout_clause ::= for time_expression
```

The `wait_statement` template is a primary template belonging to the `SEQUENTIAL_STATEMENT` class. It contains the attributes shown in [Table 104](#).

**Table 104: wait\_statement Template**

| Attribute                       | Kind     | Limit_Kind  |
|---------------------------------|----------|-------------|
| <code>cased_label</code>        | template | ID          |
| <code>label</code>              | template | ID          |
| <code>sensitivity</code>        | template | EXPRESSION  |
| <code>condition</code>          | template | EXPRESSION  |
| <code>timeout</code>            | template | EXPRESSION  |
| <code>declarative_region</code> | template | REGION_PART |
| <code>sensitivity_count</code>  | max/min  | N/A         |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```
template T1 is wait_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is wait_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end
```

The `lab1` label matches both templates, whereas the `LAB1` label only matches the first template. The `label` attribute is not case-sensitive because VHDL is not case-sensitive.

## Waveform Element Template

The LRM (§8.4.1) defines the grammar for waveform element as follows:

```
waveform_element ::=
 value_expression [after time_expression]
 | null [after time_expression]
```

The `waveform_element` template is a secondary template belonging to no classes. It contains the attributes shown in [Table 105](#). This template is used in the [Conditional Waveforms Template](#), [Selected Waveforms Template](#), and [Waveform Template](#).

**Table 105: waveform\_element Template**

| Attribute                        | Kind     | Limit_Kind |
|----------------------------------|----------|------------|
| <code>waveform_expression</code> | template | EXPRESSION |
| <code>after_expression</code>    | template | EXPRESSION |
| <code>null</code>                | local    | N/A        |

## Waveform Template

The LRM (§8.4) defines the grammar for waveform as follows:

```
waveform ::= waveform_element { , waveform_element } | unaffected
```

The waveform template is a secondary template belonging to no classes. It contains the attributes shown in [Table 106](#). This template is used in the [Signal Assignment Statement Template](#).

**Table 106: waveform Template**

| Attribute              | Kind      | Limit_Kind       |
|------------------------|-----------|------------------|
| unaffected             | local     | N/A              |
| waveform_element_s     | aggregate | waveform_element |
| waveform_element_count | max/min   | N/A              |

Use the unaffected attribute to force or forbid the presence of the unaffected VHDL keyword.

## While Loop Statement Template

The LRM (§8.9) defines the grammar for loop statement as follows:

```
loop_statement ::=
 [loop_label :]
 [iteration_scheme] loop
 sequence_of_statements
 end loop [loop_label] ;
iteration_scheme ::=
 while condition
 | for loop_parameter_specification
parameter_specification ::=
 identifier in discrete_range
```

The while\_loop\_statement template is a primary template belonging to the SEQUENTIAL\_STATEMENT class. It contains the attributes shown in [Table 107](#).

**Table 107: while\_loop\_statement Template**

| Attribute   | Kind     | Limit_Kind |
|-------------|----------|------------|
| cased_label | template | ID         |
| label       | template | ID         |
| condition   | template | EXPRESSION |

**Table 107: while\_loop\_statement Template (Continued)**

| Attribute                     | Kind      | Limit_Kind                    |
|-------------------------------|-----------|-------------------------------|
| assertion_statement           | template  | assertion_statement           |
| report_statement              | template  | report_statement              |
| procedure_call_statement      | template  | procedure_call_statement      |
| if_statement                  | template  | if_statement                  |
| signal_assignment_statement   | template  | signal_assignment_statement   |
| variable_assignment_statement | template  | variable_assignment_statement |
| case_statement                | template  | case_statement                |
| loop_statement                | template  | loop_statement                |
| for_loop_statement            | template  | for_loop_statement            |
| while_loop_statement          | template  | while_loop_statement          |
| next_statement                | template  | next_statement                |
| exit_statement                | template  | exit_statement                |
| return_statement              | template  | return_statement              |
| wait_statement                | template  | wait_statement                |
| null_statement                | local     | N/A                           |
| declarative_region            | template  | REGION_PART                   |
| statement_format              | template  | statement_format              |
| statement_profile_s           | aggregate | SEQUENTIAL_STATEMENT          |
| evaluation_time               | set       | N/A                           |

Use the `cased_label` attribute just like the `label` attribute, but in cases where the label is case-sensitive. Consider the following templates:

```

template T1 is while_loop_statement
 limit label to "^[a-z][a-z0-9_]*$"
end
template T2 is while_loop_statement
 limit cased_label to "^[a-z][a-z0-9_]*$"
end

```

The lab1 label matches both templates, whereas the LAB1 label only matches the first template. The label attribute is not case-sensitive because VHDL is not case-sensitive.





---

# A

## Template x Attribute SpecDex

---

### About the SpecDex

The Specifier Index (SpecDex) is a double cross-referencing tool that you can use to search for information in this manual about the VRSL templates and attributes. SpecDex is indexed two ways:

- **TEMPLATE x Attribute**
- **ATTRIBUTE x Template**

SpecDex is designed to be used as an online tool for developers writing VRSL code.

For example, to find all the attributes that use the `Always_construct` template, go to the **TEMPLATE x Attribute** index and scroll down until you find the `Always_construct` template. All of the attributes of the `Always_construct` template are listed by page number. Click on any attribute page number to hyperlink to the reference information about that attribute in this manual. A traditional index is also included at the end of the manual.

### **TEMPLATE x Attribute**

#### **A**

##### **Access\_type\_definition template**

subtype\_indication attribute [70](#)

##### **Aggregate template**

choice attribute [70](#)

element\_association\_s attribute [71](#)

element\_count attribute [71](#)  
multiple\_choices attribute [70](#)  
named\_association attribute [70](#)  
others attribute [71](#)  
positional\_association attribute [70](#)  
record\_aggregate attribute [70](#)

### **Alias\_declaration template**

cased\_identifier attribute [71](#)  
character\_literal attribute [71](#)  
comment attribute [71](#)  
declarative\_region attribute [71](#)  
identifier attribute [71](#)  
name attribute [71](#)  
one\_declaration\_per\_line attribute [71](#)  
operator\_symbol attribute [71](#)  
subtype\_indication attribute [71](#)

### **Allocator template**

expression attribute [72](#)  
subtype\_indication attribute [72](#)

### **Architecture\_body template**

alias\_declaration attribute [74](#)  
asynchronous\_reset\_signal attribute [75](#)  
attribute\_declaration attribute [74](#)  
attribute\_specification attribute [74](#)  
block\_statement attribute [74](#)  
cased\_identifier attribute [73](#)  
clock\_signal attribute [75](#)  
component\_declaration attribute [74](#)  
component\_instantiation\_statement attribute [74](#)

concurrent\_assertion\_statement attribute 74  
concurrent\_procedure\_call\_statement attribute 74  
conditional\_signal\_assignment attribute 74  
configuration\_specification attribute 74  
constant\_declaration attribute 74  
declaration\_profile\_s attribute 75  
design\_unit attribute 75  
disconnection\_specification attribute 74  
entity\_declaration attribute 75  
file\_declaration attribute 74  
file\_layout attribute 75  
file\_length attribute 75  
file\_name attribute 75  
fsm attribute 75  
generate\_statement attribute 74  
group\_declaration attribute 74  
group\_template\_declaration attribute 74  
header\_comment attribute 75  
identifier attribute 73  
line\_count attribute 75  
ncs\_file\_name attribute 75  
process\_statement attribute 74  
related\_unit attribute 74  
selected\_signal\_assignment attribute 74  
shared\_variable\_declaration attribute 74  
signal\_declaration attribute 74  
statement\_format attribute 74  
statement\_profile\_s attribute 75  
subprogram\_body attribute 73  
subprogram\_declaration attribute 73  
subtype\_declaration attribute 74  
synchronous\_reset\_signal attribute 75

top\_architecture attribute [75](#)

type\_declaration attribute [74](#)

use\_clause attribute [74](#)

### **Assertion\_statement template**

cased\_label attribute [77](#)

condition attribute [77](#)

declarative\_region attribute [77](#)

label attribute [77](#)

postponed attribute [77](#)

report\_expression attribute [77](#)

severity\_expression attribute [77](#)

### **Association\_element template**

actual\_designator attribute [78](#)

actual\_function attribute [78](#)

actual\_type\_mark attribute [78](#)

formal\_function attribute [78](#)

formal\_type\_mark attribute [78](#)

named\_association attribute [78](#)

open attribute [78](#)

others attribute [78](#)

positional\_association attribute [78](#)

range\_overflow attribute [78](#)

### **Association\_list template**

association\_element\_s attribute [79](#)

named\_association attribute [79](#)

open attribute [79](#)

others attribute [79](#)

parameter\_count attribute [79](#)

positional\_association attribute [79](#)

read\_write attribute [79](#)

### **Asynchronous\_initialization template**

cased\_identifier attribute [80](#)

connectivity\_path attribute [80](#)

edge attribute [80](#)

expression attribute [80](#)

gated\_in\_unit attribute [80](#)

identifier attribute [80](#)

is\_load attribute [80](#)

is\_reset attribute [80](#)

is\_set attribute [80](#)

### **Attribute\_declaration template**

cased\_identifier attribute [81](#)

comment attribute [81](#)

declarative\_region attribute [81](#)

identifier attribute [81](#)

one\_declaration\_per\_line attribute [81](#)

type\_mark attribute [81](#)

### **Attribute\_name template**

attribute\_designator attribute [82](#)

expression attribute [82](#)

name\_prefix attribute [82](#)

named\_entity\_prefix attribute [82](#)

### **Attribute\_specification template**

all attribute [83](#)

architecture attribute [83](#)

attribute\_designator attribute [83](#)

cased\_label attribute [83](#)

component attribute [84](#)  
configuration attribute [83](#)  
constant attribute [84](#)  
entity attribute [83](#)  
entity\_designator attribute [83](#)  
expression attribute [83](#)  
file attribute [84](#)  
function attribute [83](#)  
group attribute [84](#)  
label attribute [83](#)  
literal attribute [84](#)  
others attribute [83](#)  
package attribute [83](#)  
procedure attribute [83](#)  
signal attribute [84](#)  
subtype attribute [84](#)  
type attribute [84](#)  
units attribute [84](#)  
variable attribute [84](#)

## B

### **Binary\_operation template**

evaluation\_time attribute [85](#)  
left\_expression attribute [85](#)  
operand\_size\_match attribute [85](#)  
operand\_size\_mismatch attribute [85](#)  
operator\_symbol attribute [85](#)  
right\_expression attribute [85](#)  
type\_mark attribute [85](#)  
value attribute [85](#)

**Binding\_indication template**

configuration attribute [86](#)  
entity attribute [86](#)  
generic\_map\_aspect attribute [86](#)  
port\_map\_aspect attribute [86](#)

**Block\_configuration template**

block\_configuration attribute [87](#)  
block\_specification attribute [87](#)  
component\_configuration attribute [87](#)  
use\_clause attribute [87](#)

**Block\_specification template**

architecture\_name attribute [88](#)  
block\_statement\_label attribute [88](#)  
generate\_statement\_label attribute [88](#)  
index\_specification attribute [88](#)

**Block\_statement template**

alias\_declaration attribute [90](#)  
assertion\_statement attribute [90](#)  
attribute\_declaration attribute [90](#)  
attribute\_specification attribute [90](#)  
block\_statement attribute [90](#)  
cased\_label attribute [89](#)  
component\_declaration attribute [90](#)  
conditional\_signal\_assignment attribute [90](#)  
configuration\_specification attribute [90](#)  
constant\_declaration attribute [89](#)  
declaration\_profile\_s attribute [90](#)  
declarative\_region attribute [90](#)  
disconnection\_specification attribute [90](#)

file\_declaration attribute [90](#)  
generate\_statement attribute [90](#)  
generic\_declaration attribute [89](#)  
generic\_map\_aspect attribute [89](#)  
group\_declaration attribute [90](#)  
group\_template\_declaration attribute [90](#)  
guard\_expression attribute [89](#)  
label attribute [89](#)  
port\_declaration attribute [89](#)  
port\_map\_aspect attribute [89](#)  
procedure\_call\_statement attribute [90](#)  
process\_statement attribute [90](#)  
selected\_signal\_assignment attribute [90](#)  
shared\_variable\_declaration attribute [89](#)  
signal\_declaration attribute [90](#)  
statement\_format attribute [90](#)  
statement\_profile\_s attribute [90](#)  
subprogram\_body attribute [89](#)  
subprogram\_declaration attribute [89](#)  
subtype\_declaration attribute [89](#)  
type\_declaration attribute [89](#)  
use\_clause attribute [90](#)

## C

### **Case\_statement template**

alternative\_count attribute [92](#)  
assertion\_statement attribute [91](#)  
case\_statement attribute [92](#)  
cased\_label attribute [91](#)  
choice attribute [91](#)  
declarative\_region attribute [92](#)



exit\_statement attribute [92](#)  
expression attribute [91](#)  
for\_loop\_statement attribute [92](#)  
if\_statement attribute [91](#)  
label attribute [91](#)  
loop\_statement attribute [92](#)  
multiple\_choices attribute [92](#)  
next\_statement attribute [92](#)  
null\_statement attribute [92](#)  
others attribute [92](#)  
procedure\_call\_statement attribute [91](#)  
report\_statement attribute [91](#)  
return\_statement attribute [92](#)  
signal\_assignment\_statement attribute [91](#)  
statement\_format attribute [92](#)  
statement\_profile\_s attribute [92](#)  
variable\_assignment\_statement attribute [91](#)  
wait\_statement attribute [92](#)  
while\_loop\_statement attribute [92](#)

### **Clock template**

cased\_identifier attribute [93](#)  
connectivity\_path attribute [93](#)  
data attribute [93](#)  
edge attribute [93](#)  
expression attribute [93](#)  
fixed\_value attribute [93](#)  
gated\_in\_unit attribute [93](#)  
identifier attribute [93](#)  
object\_definition attribute [93](#)  
starting\_unit attribute [93](#)

### **Comment template**

enclosing\_filename attribute [94](#)

text attribute [94](#)

### **Component\_configuration template**

binding\_indication attribute [95](#)

block\_configuration attribute [95](#)

component\_specification attribute [95](#)

### **Component\_declaration template**

cased\_identifier attribute [95](#)

declarative\_region attribute [96](#)

generic\_declaration attribute [95](#)

identifier attribute [95](#)

port\_count attribute [96](#)

port\_declaration attribute [95](#)

port\_order attribute [96](#)

### **Component\_instantiation\_statement template**

cased\_label attribute [97](#)

configuration attribute [97](#)

consistent\_port\_order attribute [97](#)

db\_instantiation attribute [97](#)

declarative\_region attribute [97](#)

entity attribute [97](#)

generic\_map\_aspect attribute [97](#)

label attribute [97](#)

port\_map\_aspect attribute [97](#)

unit\_name attribute [97](#)

use\_db\_name attribute [97](#)

### **Component\_specification template**

all attribute [99](#)  
cased\_label attribute [98](#)  
label attribute [98](#)  
others attribute [99](#)  
postponed attribute [99](#)

### **Concurrent\_assertion\_statement template**

cased\_label attribute [99](#)  
condition attribute [99](#)  
declarative\_region attribute [100](#)  
label attribute [99](#)  
postponed attribute [100](#)  
report\_expression attribute [99](#)  
severity\_expression attribute [100](#)

### **Concurrent\_procedure\_call\_statement template**

actual\_parameter\_part attribute [100](#)  
cased\_identifier attribute [100](#)  
cased\_label attribute [100](#)  
declarative\_region attribute [100](#)  
identifier attribute [100](#)  
label attribute [100](#)  
postponed attribute [100](#)

### **Conditional\_signal\_assignment template**

cased\_label attribute [101](#)  
conditional\_waveforms\_s attribute [102](#)  
guarded attribute [102](#)  
inertial attribute [102](#)  
label attribute [101](#)  
postponed attribute [101](#)

range\_overflow attribute [102](#)  
read\_write attribute [102](#)  
reject\_expression attribute [101](#)  
statement\_format attribute [102](#)  
target attribute [101](#)  
transport attribute [102](#)  
waveform\_count attribute [102](#)

### **Conditional\_waveform template**

condition attribute [103](#)  
waveform attribute [103](#)

### **Configuration\_declaration template**

attribute\_specification attribute [104](#)  
block\_configuration attribute [104](#)  
cased\_identifier attribute [104](#)  
design\_unit attribute [104](#)  
file\_layout attribute [104](#)  
file\_length\_count attribute [104](#)  
file\_name attribute [104](#)  
group\_declaration attribute [104](#)  
header\_comment attribute [104](#)  
identifier attribute [104](#)  
line\_count attribute [104](#)  
ncs\_file\_name attribute [104](#)  
related\_unit attribute [104](#)  
top\_configuration attribute [104](#)  
use\_clause attribute [104](#)

### **Configuration\_specification template**

all attribute [106](#)  
binding\_indication attribute [106](#)

consistent\_port\_order attribute [106](#)  
others attribute [106](#)

### **Connectivity\_path template**

buffer\_count attribute [56](#)  
control\_src\_count attribute [56](#)  
data attribute [56](#)  
flipflop\_as\_source attribute [56](#)  
inverter\_count attribute [56](#)  
is\_combinatorial attribute [56](#)  
is\_reset attribute [56](#)  
latch\_as\_source attribute [56](#)  
starting\_unit attribute [56](#)  
within\_same\_clkdomain attribute [56](#)

### **Constant\_declaration template**

cased\_identifier attribute [106](#)  
comment attribute [107](#)  
declarative\_region attribute [107](#)  
default attribute [106](#)  
deferred attribute [107](#)  
identifier attribute [106](#)  
one\_declaration\_per\_line attribute [107](#)  
subtype\_indication attribute [106](#)

### **Constrained\_array\_definition template**

dimension\_count attribute [107](#)  
index\_constraint\_s attribute [107](#)  
subtype\_indication attribute [107](#)

## D

### Data\_signal template

connectivity\_path attribute [58](#)

fixed\_value attribute [58](#)

### Design template

asynchronous\_feedback attribute [58](#)

asynchronous\_initialization attribute [58](#)

asynchronous\_logic attribute [58](#)

clock attribute [58](#)

clock\_count attribute [59](#)

comb\_cost attribute [59](#)

drivers\_per\_signal attribute [58](#)

flipflop attribute [59](#)

gated\_clock attribute [59](#)

gated\_reset attribute [59](#)

glue\_logic\_at\_top attribute [59](#)

initialization\_count attribute [59](#)

latch attribute [59](#)

load\_count attribute [59](#)

logic\_cost attribute [59](#)

meta\_stability attribute [59](#)

mixed\_async\_sync\_line attribute [59](#)

mixed\_clock attribute [59](#)

multiplexed\_clock attribute [59](#)

non\_tristate\_drivers\_per\_signal attribute [59](#)

pulse\_generator attribute [59](#)

registered\_inputs attribute [59](#)

registered\_outputs attribute [59](#)

reset\_count attribute [59](#)

set\_count attribute [59](#)

sync\_ff\_count attribute [59](#)  
synchronous\_initialization attribute [58](#)  
top\_unit attribute [58](#)

### **Design\_unit template**

library\_clause attribute [108](#)  
library\_name attribute [108](#)  
use\_clause attribute [108](#)  
use\_work attribute [108](#)

### **Disconnection\_specification template**

all attribute [109](#)  
others attribute [109](#)  
type\_mark attribute [109](#)

## **E**

### **Element\_association template**

choice attribute [109](#)  
multiple\_choices attribute [110](#)  
others attribute [110](#)  
positional\_association attribute [109](#)  
range\_overflow attribute [109](#)

### **Element\_declaration template**

cased\_identifier attribute [110](#)  
identifier attribute [110](#)  
subtype\_indication attribute [110](#)

### **Entity\_declaration template**

alias\_declaration attribute [112](#)  
assertion\_statement attribute [112](#)

attribute\_declaration attribute [112](#)  
attribute\_specification attribute [112](#)  
cased\_identifier attribute [112](#)  
constant\_declaration attribute [112](#)  
declaration\_profile\_s attribute [113](#)  
design\_unit attribute [112](#)  
disconnection\_specification attribute [112](#)  
file\_declaration attribute [112](#)  
file\_layout attribute [113](#)  
file\_length attribute [113](#)  
file\_name attribute [113](#)  
generic\_declaration attribute [112](#)  
group\_declaration attribute [112](#)  
group\_template\_declaration attribute [112](#)  
header\_comment attribute [113](#)  
identifier attribute [112](#)  
line\_count attribute [113](#)  
ncs\_file\_name attribute [113](#)  
port\_count attribute [113](#)  
port\_declaration attribute [112](#)  
port\_order attribute [113](#)  
procedure\_call\_statement attribute [112](#)  
process\_statement attribute [112](#)  
shared\_variable\_declaration attribute [112](#)  
signal\_declaration attribute [112](#)  
subprogram\_body attribute [112](#)  
subprogram\_declaration attribute [112](#)  
subtype\_declaration attribute [112](#)  
top\_entity attribute [113](#)  
type\_declaration attribute [112](#)  
use\_clause attribute [112](#)  
use\_db\_name attribute [112](#)



**Enumeration\_type\_definition template**

cased\_identifier attribute [115](#)  
character\_literal attribute [115](#)  
element\_count attribute [115](#)  
identifier attribute [115](#)

**Exit\_statement template**

cased\_label attribute [116](#)  
condition attribute [116](#)  
declarative\_region attribute [116](#)  
label attribute [116](#)  
loop\_label attribute [116](#)

**Expression template**

evaluation\_time attribute [117](#)  
operand attribute [117](#)  
operator\_symbol [117](#)  
qualified attribute [117](#)  
type\_conversion attribute [117](#)

**F****File\_declaration template**

comment attribute [119](#)  
declarative\_region attribute [119](#)  
identifier attribute [119](#)  
logical\_name attribute [119](#)  
one\_declaration\_per\_line attribute [119](#)  
open\_kind attribute [119](#)  
subtype\_indication attribute [119](#)

**File\_layout template**

cased\_identifier attribute [120](#)  
characters\_per\_line attribute [120](#)  
file\_length attribute [120](#)  
file\_name attribute [120](#)  
header\_comment attribute [120](#)  
identifier attribute [120](#)  
related\_units attribute [120](#)  
unit\_count attribute [120](#)

**File\_type\_definition template**

type\_mark attribute [121](#)

**Flipflop template**

asynchronous\_initialization attribute [121](#)  
cased\_identifier attribute [60](#), [121](#)  
clock attribute [61](#), [122](#)  
data\_signal attribute [61](#), [122](#)  
has\_clock\_as\_data attribute [61](#), [122](#)  
identifier attribute [60](#), [121](#)  
input attribute [60](#), [121](#)  
object\_definition attribute [60](#), [122](#)  
output attribute [60](#), [121](#), [122](#)

**Floating\_type\_definition template**

range attribute [122](#)

**For\_loop\_statement template**

assertion\_statement attribute [123](#)  
case\_statement attribute [123](#)  
cased\_label attribute [123](#)  
declarative\_region attribute [124](#)

evaluation\_time attribute [124](#)  
exit\_statement attribute [123](#)  
for\_loop\_statement attribute [123](#)  
if\_statement attribute [123](#)  
label attribute [123](#)  
loop\_statement attribute [123](#)  
next\_statement attribute [123](#)  
null\_statement attribute [124](#)  
procedure\_call\_statement attribute [123](#)  
report\_statement attribute [123](#)  
return\_statement attribute [123](#)  
signal\_assignment\_statement attribute [123](#)  
statement\_format attribute [124](#)  
statement\_profile\_s attribute [124](#)  
subtype\_indication attribute [123](#)  
variable\_assignment\_statement attribute [123](#)  
wait\_statement attribute [124](#)  
while\_loop\_statement attribute [123](#)

### **Formal\_parameter template**

cased\_identifier attribute [125](#)  
comment attribute [125](#)  
constant attribute [125](#)  
default attribute [125](#)  
file attribute [125](#)  
identifier attribute [125](#)  
in attribute [125](#)  
inout attribute [125](#)  
one\_declaration\_per\_line attribute [125](#)  
out attribute [125](#)  
signal attribute [125](#)  
subtype\_indication attribute [125](#)

variable attribute [125](#)

### **Fsm template**

block\_count attribute [118](#)

mealy attribute [118](#)

moore attribute [118](#)

state\_count attribute [118](#)

state\_variable attribute [118](#)

transition\_in\_default attribute [118](#)

### **Function\_call template**

actual\_parameter\_part attribute [126](#)

cased\_identifier attribute [126](#)

identifier attribute [126](#)

## **G**

### **Generate\_statement template**

alias\_declaration attribute [128](#)

assertion\_statement attribute [129](#)

attribute\_declaration attribute [128](#)

attribute\_specification attribute [129](#)

cased\_label attribute [128](#)

component\_declaration attribute [128](#)

component\_instantiation\_statement attribute [129](#)

condition attribute [128](#)

conditional\_signal\_assignment attribute [129](#)

configuration\_specification attribute [129](#)

constant\_declaration attribute [128](#)

declarative\_region attribute [129](#)

disconnection\_specification attribute [129](#)

discrete\_range attribute [128](#)

file\_declaration attribute [128](#)  
generate\_statement attribute [129](#)  
group\_declaration attribute [129](#)  
group\_template\_declaration attribute [129](#)  
procedure\_call\_statement attribute [129](#)  
process\_statement attribute [129](#)  
selected\_signal\_assignment attribute [129](#)  
shared\_variable\_declaration attribute [128](#)  
signal\_declaration attribute [128](#)  
subprogram\_body attribute [128](#)  
subprogram\_declaration attribute [128](#)  
subtype\_declaration attribute [128](#)  
type\_declaration attribute [128](#)  
use\_clause attribute [129](#)

### **Generic\_declaration template**

cased\_identifier attribute [130](#)  
comment attribute [130](#)  
constant attribute [130](#)  
declarative\_region attribute [130](#)  
default attribute [130](#)  
identifier attribute [130](#)  
in attribute [130](#)  
one\_declaration\_per\_line attribute [130](#)  
subtype\_indication attribute [130](#)

## **H**

### **Header\_comment template**

comment\_line attribute [132](#)

## I

### Identifier template

character\_count attribute [133](#)

error\_id attribute [133](#)

limit\_id attribute [133](#)

### If\_statement template

assertion\_statement attribute [134](#)

case\_statement attribute [134](#)

cased\_label attribute [134](#)

condition\_count attribute [135](#)

condition\_s attribute [135](#)

declarative\_region attribute [135](#)

else attribute [135](#)

exit\_statement attribute [134](#)

for\_loop\_statement attribute [134](#)

if\_statement attribute [134](#)

indentation\_depth attribute [134](#)

label attribute [134](#)

loop\_statement attribute [134](#)

next\_statement attribute [134](#)

null\_statement attribute [135](#)

procedure\_call\_statement attribute [134](#)

report\_statement attribute [134](#)

return\_statement attribute [134](#)

signal\_assignment\_statement attribute [134](#)

statement\_format attribute [135](#)

statement\_profile\_s attribute [135](#)

variable\_assignment\_statement attribute [134](#)

wait\_statement attribute [134](#)

while\_loop\_statement attribute [134](#)

### **Indexed\_name template**

cased\_identifier attribute [136](#)  
expression attribute [136](#)  
flipflop attribute [136](#)  
identifier attribute [136](#)  
latch attribute [136](#)  
object\_definition attribute [136](#)  
range\_overflow attribute [136](#)

### **Integer\_type\_definition template**

range attribute [137](#)

### **Interface\_file\_declaration template**

cased\_identifier attribute [138](#)  
comment attribute [138](#)  
declarative\_region attribute [138](#)  
file attribute [138](#)  
identifier attribute [138](#)  
one\_declaration\_per\_line attribute [138](#)  
subtype\_indication attribute [138](#)

### **Interface\_variable\_declaration template**

cased\_identifier attribute [139](#)  
comment attribute [139](#)  
declarative\_region attribute [139](#)  
default attribute [139](#)  
identifier attribute [139](#)  
in attribute [139](#)  
inout attribute [139](#)  
linkage attribute [139](#)  
one\_declaration\_per\_line attribute [139](#)  
out attribute [139](#)

subtype\_indication attribute [139](#)  
variable attribute [139](#)

## L

### Latch template

asynchronous\_initialization attribute [62](#), [140](#)  
cased\_identifier attribute [62](#), [140](#)  
clock attribute [62](#), [140](#)  
data\_signal attribute [62](#), [140](#)  
has\_clock\_as\_data attribute [62](#), [140](#)  
identifier attribute [62](#), [140](#)  
input attribute [62](#), [140](#)  
object\_definition attribute [62](#)  
output attribute [62](#), [140](#)  
synchronous\_initialization attribute [62](#), [140](#)

### Literal template

base attribute [141](#)  
base\_specifier attribute [141](#)  
evaluation\_time attribute [141](#)  
null attribute [141](#)  
use\_exponent attribute [141](#)  
value attribute [141](#)  
value\_type attribute [141](#)

### Logic\_cost template

abs\_cost attribute [63](#)  
and\_cost attribute [63](#)  
buffer\_cost attribute [63](#)  
comparator\_cost attribute [63](#)  
decoder\_cost attribute [63](#)



divide\_cost attribute [63](#)  
function\_cost attribute [63](#)  
max\_cost attribute [63](#)  
minus\_cost attribute [63](#)  
modulus\_cost attribute [63](#)  
multiply\_cost attribute [63](#)  
mux\_cost attribute [63](#)  
nand\_cost attribute [63](#)  
nor\_cost attribute [63](#)  
or\_cost attribute [63](#)  
plus\_cost attribute [63](#)  
power\_cost attribute [63](#)  
remainder\_cost attribute [63](#)  
reset\_at\_hierarchical\_boundary attribute [63](#)  
shift\_cost attribute [63](#)  
xnor\_cost attribute [63](#)  
xor\_cost attribute [63](#)

### **Loop\_statement template**

assertion\_statement attribute [143](#)  
case\_statement attribute [143](#)  
cased\_label attribute [143](#)  
declarative\_region attribute [144](#)  
exit\_statement attribute [143](#)  
for\_loop\_statement attribute [143](#)  
if\_statement attribute [143](#)  
label attribute [143](#)  
loop\_statement attribute [143](#)  
next\_statement attribute [143](#)  
null\_statement attribute [144](#)  
procedure\_call\_statement attribute [143](#)  
report\_statement attribute [143](#)

return\_statement attribute [143](#)  
signal\_assignment\_statement attribute [143](#)  
statement\_format attribute [144](#)  
statement\_profile\_s attribute [144](#)  
variable\_assignment\_statement attribute [143](#)  
wait\_statement attribute [143](#)  
while\_loop\_statement attribute [143](#)

## N

### **Next\_statement template**

cased\_label attribute [145](#)  
condition attribute [145](#)  
declarative\_region attribute [145](#)  
label attribute [145](#)  
loop\_label attribute [145](#)

## P

### **Package\_body template**

alias\_declaration attribute [146](#)  
constant\_declaration attribute [146](#)  
design\_unit attribute [147](#)  
file\_declaration attribute [146](#)  
file\_layout attribute [147](#)  
file\_length attribute [147](#)  
file\_name attribute [147](#)  
group\_declaration attribute [147](#)  
group\_template\_declaration attribute [146](#)  
header\_comment attribute [147](#)  
line\_count attribute [147](#)  
ncs\_file\_name attribute [147](#)

shared\_variable\_declaration attribute 146  
statement\_format attribute 147  
subprogram\_body attribute 146  
subprogram\_declaration attribute 146  
subtype\_declaration attribute 146  
type\_declaration attribute 146  
use\_clause attribute 146

### **Package\_declaration template**

alias\_declaration attribute 149  
attribute\_declaration attribute 149  
attribute\_specification attribute 149  
cased\_identifier attribute 148  
component\_declaration attribute 149  
constant\_declaration attribute 148  
declaration\_profile\_s attribute 149  
design\_unit attribute 149  
disconnection\_specification attribute 149  
file\_declaration attribute 149  
file\_layout attribute 149  
file\_length attribute 149  
file\_name attribute 149  
group\_declaration attribute 149  
group\_template\_declaration attribute 149  
header\_comment attribute 149  
identifier attribute 148  
line\_count attribute 149  
ncs\_file\_name attribute 149  
shared\_variable attribute 148  
signal\_declaration attribute 149  
statement\_format attribute 149  
subprogram\_declaration attribute 148

subtype\_declaration attribute [148](#)  
type\_declaration attribute [148](#)  
use\_clause attribute [149](#)

### **Physical\_type\_definition template**

cased\_identifier attribute [150](#)  
identifier attribute [150](#)  
range attribute [150](#)

### **Port\_declaration template**

buffer attribute [151](#)  
bus attribute [151](#)  
cased\_identifier attribute [151](#)  
comment attribute [152](#)  
connections attribute [152](#)  
declarative\_region attribute [151](#)  
default attribute [151](#)  
driving\_expression attribute [152](#)  
flipflop attribute [152](#)  
identifier attribute [151](#)  
in attribute [151](#)  
inout attribute [151](#)  
is\_clock attribute [151](#)  
is\_reset attribute [151](#)  
latch attribute [152](#)  
linkage attribute [151](#)  
one\_declaration\_per\_line attribute [152](#)  
out attribute [151](#)  
outputs\_driven attribute [152](#)  
register attribute [152](#)  
signal attribute [151](#)  
signals\_driven attribute [151](#)

subtype\_indication attribute [151](#)  
tristate attribute [152](#)

### **Procedure\_call\_statement template**

actual\_parameter\_part attribute [154](#)  
cased\_identifier attribute [154](#)  
declarative\_region attribute [154](#)  
identifier attribute [154](#)

### **Process\_statement template**

alias\_declaration attribute [156](#)  
assertion\_statement attribute [156](#)  
asynchronous\_reset attribute [157](#)  
attribute\_declaration attribute [156](#)  
attribute\_specification attribute [156](#)  
case\_statement attribute [156](#)  
cased\_label attribute [155](#)  
clock\_expression\_count attribute [157](#)  
clock\_signal attribute [157](#)  
combinatorial attribute [157](#)  
complete\_sensitivity attribute [157](#)  
constant\_declaration attribute [155](#)  
dead\_code attribute [157](#)  
declarative\_region attribute [157](#)  
exit\_statement attribute [156](#)  
file\_declaration attribute [156](#)  
flipflop attribute [157](#)  
for\_loop\_statement attribute [156](#)  
fsm attribute [157](#)  
fully\_assign\_signals attribute [157](#)  
fully\_assign\_variables attribute [157](#)  
group\_declaration attribute [156](#)

group\_template\_declaration attribute [156](#)  
header\_comment attribute [157](#)  
if\_statement attribute [156](#)  
initialize\_signals attribute [157](#)  
initialize\_variables attribute [157](#)  
label attribute [155](#)  
latch attribute [157](#)  
loop\_statement attribute [156](#)  
maximum\_variable\_usage attribute [157](#)  
missing\_signals\_in\_sensitivity\_list attribute [157](#)  
missing\_signals\_jn\_sensitivity\_list attribute [157](#)  
next\_statement attribute [156](#)  
null\_statement attribute [156](#)  
postponed attribute [156](#)  
procedure\_call\_statement attribute [156](#)  
redundancy\_in\_sensitivity\_list attribute [157](#)  
report\_statement attribute [156](#)  
return\_statement attribute [156](#)  
sensitivity attribute [155](#)  
sensitivity\_count attribute [157](#)  
signal\_assignment\_statement attribute [156](#)  
statement\_format attribute [157](#)  
statement\_profile\_s attribute [157](#)  
subprogram\_body attribute [155](#)  
subprogram\_declaration attribute [155](#)  
subtype\_declaration attribute [155](#)  
synchronous\_reset attribute [157](#)  
type\_declaration attribute [155](#)  
unused\_declaration attribute [156](#)  
use\_clause attribute [156](#)  
variable\_assignment\_statement attribute [156](#)  
variable\_declaration attribute [156](#)

wait\_statement attribute [156](#)  
while\_loop\_statement attribute [156](#)

## R

### **Range template**

downto attribute [159](#)  
evaluation\_time attribute [159](#)  
high\_bound attribute [159](#)  
left\_expression attribute [159](#)  
low\_bound attribute [159](#)  
null\_range attribute [159](#)  
range\_attribute attribute [159](#)  
range\_index attribute [159](#)  
right\_expression attribute [159](#)  
to attribute [159](#)

### **Record\_type\_definition template**

element\_declaration attribute [161](#)  
type\_mark attribute [161](#)

### **Report\_statement template**

cased\_label attribute [161](#)  
declarative\_region attribute [161](#)  
label attribute [161](#)  
report\_expression attribute [161](#)  
severity\_expression attribute [161](#)

### **Return\_statement template**

cased\_label attribute [162](#)  
declarative\_region attribute [162](#)  
expression attribute [162](#)

label attribute [162](#)

## S

### **Selected\_name template**

cased\_identifier attribute [163](#)

identifier attribute [163](#)

object\_definition attribute [163](#)

### **Selected\_signal\_assignment template**

cased\_label attribute [164](#)

expression attribute [164](#)

guarded attribute [164](#)

inertial attribute [164](#)

label attribute [164](#)

operand\_size\_mismatch attribute [164](#)

postponed attribute [164](#)

range\_overflow attribute [164](#)

read\_write attribute [164](#)

reject\_expression attribute [164](#)

selected\_waveform\_s attribute [164](#)

statement\_format attribute [164](#)

target attribute [164](#)

transport attribute [164](#)

waveform\_count attribute [164](#)

### **Selected\_waveforms template**

choice attribute [165](#)

others attribute [165](#)

waveform attribute [165](#)



### **Shared\_variable\_declaration template**

cased\_identifier attribute [166](#)  
comment attribute [166](#)  
declarative\_region attribute [166](#)  
default attribute [166](#)  
identifier attribute [166](#)  
one\_declaration\_per\_line attribute [166](#)  
subtype\_indication attribute [166](#)

### **Signal\_assignment\_statement template**

cased\_label attribute [167](#)  
declarative\_region attribute [167](#)  
inertial attribute [167](#)  
label attribute [167](#)  
operand\_size\_mismatch attribute [167](#)  
range\_overflow attribute [167](#)  
read\_write attribute [167](#)  
reject\_expression attribute [167](#)  
target attribute [167](#)  
transport attribute [167](#)  
waveform attribute [167](#)  
waveform\_count attribute [167](#)

### **Signal\_declaration template**

bus attribute [168](#)  
cased\_identifier attribute [168](#)  
comment attribute [169](#)  
declarative\_region attribute [168](#)  
default attribute [168](#)  
driving\_expression attribute [168](#)  
flipflop attribute [168](#)  
identifier attribute [168](#)

is\_clock attribute [169](#)  
is\_reset attribute [169](#)  
latch attribute [168](#)  
one\_declaration\_per\_line attribute [169](#)  
outputs\_driven attribute [168](#)  
register attribute [168](#)  
signals\_driven attribute [168](#)  
subtype\_indication attribute [168](#)  
tristate attribute [169](#)

### **Simple\_name template**

cased\_identifier attribute [170](#)  
flipflop attribute [170](#)  
identifier attribute [170](#)  
latch attribute [170](#)  
object\_definition attribute [170](#)

### **Slice\_name template**

cased\_identifier attribute [171](#)  
flipflop attribute [171](#)  
identifier attribute [171](#)  
latch attribute [171](#)  
object\_definition attribute [171](#)  
range\_overflow attribute [171](#)  
ranger attribute [171](#)

### **Statement\_format template**

end\_label attribute [172](#)  
indentation\_depth attribute [172](#)  
line\_count attribute [172](#)  
one\_statement\_per\_line attribute [172](#)

## **Subprogram\_body template**

alias\_declaration attribute [174](#)  
assertion\_statement attribute [174](#)  
attribute\_declaration attribute [174](#)  
attribute\_specification attribute [174](#)  
case\_statement attribute [175](#)  
clock\_expression\_count attribute [175](#)  
clock\_signal attribute [175](#)  
constant\_declaration attribute [174](#)  
dead\_code attribute [175](#)  
exit\_statement attribute [175](#)  
file\_declaration attribute [174](#)  
for\_loop\_statement attribute [175](#)  
fully\_assign\_variables attribute [174](#)  
group\_declaration attribute [174](#)  
group\_template\_declaration attribute [174](#)  
header\_comment attribute [175](#)  
if\_statement attribute [174](#)  
loop\_statement attribute [175](#)  
next\_statement attribute [175](#)  
null\_statement attribute [175](#)  
out\_params\_fully\_assigned attribute [175](#)  
procedure\_call\_statement attribute [174](#)  
recursion\_type attribute [175](#)  
report\_statement attribute [174](#)  
return\_last attribute [175](#)  
return\_statement attribute [175](#)  
signal\_assignment\_statement attribute [174](#)  
statement\_format attribute [175](#)  
statement\_profile\_s attribute [175](#)  
subprogram\_body attribute [174](#)  
subprogram\_declaration attribute [174](#)

subtype\_declaration attribute [174](#)  
type\_declaration attribute [174](#)  
use\_clause attribute [174](#)  
variable\_assignment\_statement attribute [174](#)  
variable\_declaration attribute [174](#)  
wait\_statement attribute [175](#)  
while\_loop\_statement attribute [175](#)

### **Subprogram\_declaration template**

cased\_identifier attribute [178](#)  
declarative\_region attribute [178](#)  
formal\_parameter attribute [178](#)  
function attribute [178](#)  
header\_comment attribute [178](#)  
identifier attribute [178](#)  
operator\_symbol attribute [178](#)  
procedure attribute [178](#)  
pure attribute [178](#)  
subprogram\_body attribute [178](#)  
type\_mark attribute [178](#)

### **Subtype\_declaration template**

cased\_identifier attribute [180](#)  
declarative\_region attribute [180](#)  
identifier attribute [180](#)

### **Subtype\_indication template**

evaluation\_time attribute [181](#)  
range attribute [181](#)  
resolution\_function attribute [181](#)  
subtype\_indication attribute [181](#)  
type\_mark attribute [181](#)

### **Synchronous\_initialization template**

cased\_identifier attribute [182](#)  
connectivity\_path attribute [182](#)  
edge attribute [182](#)  
expression attribute [182](#)  
gated\_in\_unit attribute [182](#)  
identifier attribute [182](#)  
is\_load attribute [182](#)  
is\_reset attribute [182](#)  
is\_set attribute [182](#)  
object\_definition attribute [182](#)

## **T**

### **Test\_signal template**

control\_at\_start attribute [65](#)  
disable\_control attribute [65](#)  
hold\_latch\_data attribute [65](#)  
reach\_memory attribute [65](#)

### **Type\_declaration template**

cased\_identifier attribute [183](#)  
declarative\_region attribute [183](#)  
identifier attribute [183](#)  
incomplete\_type\_declaration attribute [183](#)

## **U**

### **Unconstrained\_array\_definition template**

dimension\_count attribute [184](#)  
subtype\_definition attribute [184](#)  
subtype\_indication attribute [184](#)

**Use\_clause template**

selected\_name attribute [185](#)

**V****Variable\_assignment\_statement template**

cased\_label attribute [185](#)

declarative\_region attribute [185](#)

expression attribute [185](#)

label attribute [185](#)

operand\_size\_mismatch attribute [185](#)

range\_overflow attribute [185](#)

read\_write attribute [185](#)

target attribute [185](#)

**Variable\_declaration template**

cased\_identifier attribute [186](#)

comment attribute [186](#)

declarative\_region attribute [186](#)

default attribute [186](#)

identifier attribute [186](#)

one\_declaration\_per\_line attribute [186](#)

subtype\_indication attribute [186](#)

**W****Wait\_statement template**

cased\_label attribute [187](#)

condition attribute [187](#)

declarative\_region attribute [187](#)

label attribute [187](#)

sensitivity attribute [187](#)

sensitivity\_count attribute [187](#)  
timeout attribute [187](#)

### **Waveform template**

unaffected attribute [189](#)  
waveform\_element\_count attribute [189](#)  
waveform\_element\_s attribute [189](#)

### **Waveform\_element template**

after\_expression attribute [188](#)  
null attribute [188](#)  
waveform\_expression attribute [188](#)

### **While\_loop\_statement template**

assertion\_statement attribute [190](#)  
case\_statement attribute [190](#)  
cased\_label attribute [189](#)  
condition attribute [189](#)  
declarative\_region attribute [190](#)  
evaluation\_time attribute [190](#)  
exit\_statement attribute [190](#)  
for\_loop\_statement attribute [190](#)  
if\_statement attribute [190](#)  
label attribute [189](#)  
loop\_statement attribute [190](#)  
next\_statement attribute [190](#)  
null\_statement attribute [190](#)  
procedure\_call\_statement attribute [190](#)  
report\_statement attribute [190](#)  
return\_statement attribute [190](#)  
signal\_assignment\_statement attribute [190](#)  
statement\_format attribute [190](#)

statement\_profile\_s attribute [190](#)  
variable\_assignment\_statement attribute [190](#)  
wait\_statement attribute [190](#)  
while\_loop\_statement attribute [190](#)



---

**B**

---

# Attribute x Template SpecDex

---

## About the SpecDex

The Specifier Index (SpecDex) is a double cross-referencing tool that you can use to search for information in this manual about the VRSL templates and attributes. SpecDex is indexed two ways:

- **TEMPLATE x Attribute**
- **ATTRIBUTE x Template**

SpecDex is designed to be used as an online tool for developers writing VRSL code.

For example, to find all the templates that use the `file_name` attribute, go to the **ATTRIBUTE x Template** index and scroll down until you find the `file_name` attribute. All of the templates in which `file_name` occurs are listed by page number. Click on any template page number to hyperlink to the reference information about that attribute in this manual. A traditional index is also included at the end of the manual.

## **ATTRIBUTE x Template**

### **A**

#### **abs\_cost attribute**

Logic\_cost template [63](#)

#### **actual\_designator attribute**

Association\_element template [78](#)

**actual\_function attribute**

Association\_element template [78](#)

**actual\_parameter\_part attribute**

Concurrent\_procedure\_call\_statement template [100](#)

Function\_call template [126](#)

Procedure\_call\_statement template [154](#)

**actual\_type\_mark attribute**

Association\_element template [78](#)

**after\_expression attribute**

Waveform\_element template [188](#)

**alias\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

Package\_declaration template [149](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

**all attribute**

Attribute\_specification template [83](#)

Component\_specification template [99](#)

Configuration\_specification template [106](#)

Disconnection\_specification template [109](#)

**alternative\_count attribute**

Case\_statement template [92](#)

**and\_cost attribute**

Logic\_cost template [63](#)

**architecture attribute**

Attribute\_specification template [83](#)

**architecture\_name attribute**

Block\_specification template [88](#)

**assertion\_statement attribute**

Block\_statement template [90](#)

Case\_statement template [91](#)

Entity\_declaration template [112](#)

For\_loop\_statement template [123](#)

Generate\_statement template [129](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

While\_loop\_statement template [190](#)

**association\_element\_s attribute**

Association\_list template [79](#)

**asynchronous\_feedback attribute**

Design template [58](#)

**asynchronous\_initialization attribute**

Design template [58](#)

Flipflop template [121](#)

Latch template [62](#), [140](#)

**asynchronous\_logic attribute**Design template [58](#)**asynchronous\_reset attribute**Process\_statement template [157](#)**asynchronous\_reset\_signal attribute**Architecture\_body template [75](#)**attribute\_declaration attribute**Architecture\_body template [74](#)Block\_statement template [90](#)Entity\_declaration template [112](#)Generate\_statement template [128](#)Package\_declaration template [149](#)Process\_statement template [156](#)Subprogram\_body template [174](#)**attribute\_designator attribute**Attribute\_name template [82](#)Attribute\_specification template [83](#)**attribute\_specification attribute**Architecture\_body template [74](#)Block\_statement template [90](#)Configuration\_declaration template [104](#)Entity\_declaration template [112](#)Generate\_statement template [129](#)Package\_declaration template [149](#)Process\_statement template [156](#)Subprogram\_body template [174](#)

## B

### **base attribute**

Literal template [141](#)

### **base\_specifier attribute**

Literal template [141](#)

### **binding\_indication attribute**

Component\_configuration template [95](#)

Configuration\_specification template [106](#)

### **block\_configuration attribute**

Block\_configuration template [87](#)

Component\_configuration template [95](#)

Configuration\_declaration template [104](#)

### **block\_count attribute**

Fsm template [118](#)

### **block\_specification attribute**

Block\_configuration template [87](#)

### **block\_statement attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Generate\_statement template [129](#)

### **block\_statement\_label attribute**

Block\_specification template [88](#)

**buffer attribute**

Interface\_variable\_declaration template [139](#)

Port\_declaration template [151](#)

**buffer\_cost attribute**

Logic\_cost template [63](#)

**buffer\_count attribute**

Connectivity\_path template [56](#)

**bus attribute**

Port\_declaration template [151](#)

Signal\_declaration template [168](#)

**C****case\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

**cased\_identifier attribute**

Alias\_declaration template [71](#)

Architecture\_body template [73](#)

Asynchronous\_initialization template [80](#)

Attribute\_declaration template [81](#)

Clock template [93](#)

Component\_declaration template [95](#)

Concurrent\_procedure\_call\_statement template [100](#)  
Configuration\_declaration template [104](#)  
Constant\_declaration template [106](#)  
Element\_declaration template [110](#)  
Entity\_declaration template [112](#)  
Enumeration\_type\_definition template [115](#)  
File\_declaration template [119](#)  
File\_layout template [120](#)  
Flipflop template [60](#), [121](#)  
Formal\_parameter template [125](#)  
Function\_call template [126](#)  
Generic\_declaration template [130](#)  
Indexed\_name template [136](#)  
Interface\_file\_declaration template [138](#)  
Interface\_variable\_declaration template [139](#)  
Latch template [62](#), [140](#)  
Package\_declaration template [148](#)  
Physical\_type\_definition template [150](#)  
Port\_declaration template [151](#)  
Procedure\_call\_statement template [154](#)  
Selected\_name template [163](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [168](#)  
Simple\_name template [170](#)  
Slice\_name template [171](#)  
Subprogram\_declaration template [178](#)  
Subtype\_declaration template [180](#)  
Synchronous\_initialization template [182](#)  
Type\_declaration template [183](#)  
Variable\_declaration template [186](#)

**cased\_label attribute**

- Assertion\_statement template [77](#)
- Attribute\_specification template [83](#)
- Block\_statement template [89](#)
- Case\_statement template [91](#)
- Component\_instantiation\_statement template [97](#)
- Component\_specification template [98](#)
- Concurrent\_assertion\_statement template [99](#)
- Concurrent\_procedure\_call\_statement template [100](#)
- Conditional\_signal\_assignment template [101](#)
- Exit\_statement template [116](#)
- For\_loop\_statement template [123](#)
- Generate\_statement template [128](#)
- If\_statement template [134](#)
- Loop\_statement template [143](#)
- Next\_statement template [145](#)
- Process\_statement template [155](#)
- Report\_statement template [161](#)
- Return\_statement template [162](#)
- Selected\_signal\_assignment template [164](#)
- Signal\_assignment\_statement template [167](#)
- Variable\_assignment\_statement template [185](#)
- Wait\_statement template [187](#)
- While\_loop\_statement template [189](#)

**character\_count attribute**

- Identifier template [133](#)

**character\_literal attribute**

- Alias\_declaration template [71](#)
- Enumeration\_type\_definition template [115](#)



**characters\_per\_line attribute**

File\_layout template [120](#)

**choice attribute**

Aggregate template [70](#)

Case\_statement template [91](#)

Element\_association template [109](#)

Selected\_waveforms template [165](#)

**clock attribute**

Design template [58](#)

Flipflop template [61](#), [122](#)

Latch template [62](#), [140](#)

**clock\_count attribute**

Design template [59](#)

**clock\_expression\_count attribute**

Process\_statement template [157](#)

Subprogram\_body template [175](#)

**clock\_signal attribute**

Architecture\_body template [75](#)

Process\_statement template [157](#)

Subprogram\_body template [175](#)

**comb\_cost attribute**

Design template [59](#)

**combinatorial attribute**

Process\_statement template [157](#)

**comment attribute**

Alias\_declaration template [71](#)  
Attribute\_declaration template [81](#)  
Constant\_declaration template [107](#)  
File\_declaration template [119](#)  
Generic\_declaration template [130](#)  
Interface\_file\_declaration template [138](#)  
Interface\_variable\_declaration template [139](#)  
Port\_declaration template [152](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [169](#)  
Variable\_declaration template [186](#)

**comment\_line attribute**

Header\_comment template [132](#)

**comparator\_cost attribute**

Logic\_cost template [63](#)

**complete\_sensitivity attribute**

Process\_statement template [157](#)

**component attribute**

Attribute\_specification template [84](#)

**component\_configuration attribute**

Block\_configuration template [87](#)

**component\_declaration attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Generate\_statement template [128](#)

Package\_declaration template [149](#)

### **component\_instantiation\_statement attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Generate\_statement template [129](#)

### **component\_specification attribute**

Component\_configuration template [95](#)

### **concurrent\_assertion\_statement attribute**

Architecture\_body template [74](#)

### **concurrent\_procedure\_call\_statement attribute**

Architecture\_body template [74](#)

### **condition attribute**

Assertion\_statement template [77](#)

Concurrent\_assertion\_statement template [99](#)

Conditional\_waveform template [103](#)

Exit\_statement template [116](#)

Generate\_statement template [128](#)

Next\_statement template [145](#)

Wait\_statement template [187](#)

While\_loop\_statement template [189](#)

### **condition\_count attribute**

If\_statement template [135](#)

### **condition\_s attribute**

If\_statement template [135](#)

**conditional\_signal\_assignment attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Generate\_statement template [129](#)

**conditional\_waveforms\_s attribute**

Conditional\_signal\_assignment template [102](#)

**configuration attribute**

Attribute\_specification template [83](#)  
Binding\_indication template [86](#)  
Component\_instantiation\_statement template [97](#)

**configuration\_specification attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Generate\_statement template [129](#)

**connections attribute**

Port\_declaration template [152](#)

**connectivity\_path attribute**

Asynchronous\_initialization template [80](#)  
Clock template [93](#)  
Data\_signal template [58](#)  
Synchronous\_initialization template [182](#)

**consistent\_port\_order attribute**

Configuration\_specification template [106](#)

**constant attribute**

Attribute\_specification template [84](#)

Formal\_parameter template [125](#)  
Generic\_declaration template [130](#)

### **constant\_declaration attribute**

Architecture\_body template [74](#)  
Block\_statement template [89](#)  
Entity\_declaration template [112](#)  
Generate\_statement template [128](#)  
Package\_body template [146](#)  
Package\_declaration template [148](#)  
Process\_statement template [155](#)  
Subprogram\_body template [174](#)

### **control\_at\_start attribute**

Test\_signal template [65](#)

### **control\_src\_count attribute**

Connectivity\_path template [56](#)

## **D**

### **data attribute**

Clock template [93](#)  
Connectivity\_path template [56](#)

### **data\_signal attribute**

Flipflop template [61](#), [122](#)  
Latch template [62](#), [140](#)

### **dead\_code attribute**

Process\_statement template [157](#)  
Subprogram\_body template [175](#)

**declaration\_profile\_s attribute**

Block\_statement template [90](#)  
Entity\_declaration template [113](#)  
Package\_declaration template [149](#)  
Package\_declaration template [149](#)

**declarative\_region attribute**

Alias\_declaration template [71](#)  
Assertion\_statement template [77](#)  
Attribute\_declaration template [81](#)  
Block\_statement template [90](#)  
Case\_statement template [92](#)  
Component\_declaration template [96](#)  
Component\_instantiation\_statement template [97](#)  
Concurrent\_assertion\_statement template [100](#)  
Concurrent\_procedure\_call\_statement template [100](#)  
Constant\_declaration template [107](#)  
Exit\_statement template [116](#)  
File\_declaration template [119](#)  
For\_loop\_statement template [124](#)  
Generate\_statement template [129](#)  
Generic\_declaration template [130](#)  
If\_statement template [135](#)  
Interface\_file\_declaration template [138](#)  
Interface\_variable\_declaration template [139](#)  
Loop\_statement template [144](#)  
Next\_statement template [145](#)  
Port\_declaration template [151](#)  
Procedure\_call\_statement template [154](#)  
Process\_statement template [157](#)  
Report\_statement template [161](#)  
Return\_statement template [162](#)

Shared\_variable\_declaration template [166](#)  
Signal\_assignment\_statement template [167](#)  
Signal\_declaration template [168](#)  
Subprogram\_declaration template [178](#)  
Subtype\_declaration template [180](#)  
Type\_declaration template [183](#)  
Variable\_assignment\_statement template [185](#)  
Variable\_assignment\_statement template [185](#)  
Variable\_declaration template [186](#)  
Wait\_statement template [187](#)  
While\_loop\_statement template [190](#)

### **decoder\_cost attribute**

Logic\_cost template [63](#)

### **default attribute**

Constant\_declaration template [106](#)  
Formal\_parameter template [125](#)  
Generic\_declaration template [130](#)  
Interface\_variable\_declaration template [139](#)  
Port\_declaration template [151](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [168](#)  
Variable\_declaration template [186](#)

### **deferred attribute**

Constant\_declaration template [107](#)

### **design\_unit attribute**

Architecture\_body template [75](#)  
Configuration\_declaration template [104](#)  
Entity\_declaration template [112](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

### **dimension\_count attribute**

Constrained\_array\_definition template [107](#)

Unconstrained\_array\_definition template [184](#)

### **disable\_control attribute**

Test\_signal template [65](#)

### **disconnection\_specification attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Entity\_declaration template [112](#)

Generate\_statement template [129](#)

Package\_declaration template [149](#)

### **discrete\_range attribute**

Generate\_statement template [128](#)

### **divide\_cost attribute**

Logic\_cost template [63](#)

### **downto attribute**

Range template [159](#)

### **drivers\_per\_signal attribute**

Design template [58](#)

### **driving\_expression attribute**

Port\_declaration template [152](#)

Signal\_declaration template [168](#)



## E

### **edge attribute**

Asynchronous\_initialization template [80](#)

Clock template [93](#)

Synchronous\_initialization template [182](#)

### **element\_association\_s attribute**

Aggregate template [71](#)

### **element\_count attribute**

Aggregate template [71](#)

Enumeration\_type\_definition template [115](#)

### **element\_declaration attribute**

Record\_type\_definition template [161](#)

### **else attribute**

If\_statement template [135](#)

### **enclosing\_filename attribute**

Comment template [94](#)

### **end\_label attribute**

Statement\_format template [172](#)

### **entity attribute**

Attribute specification template [83](#)

Attribute\_specification template [83](#)

Binding\_indication template [86](#)

Component\_instantiation\_statement template [97](#)

**entity\_aspect\_name attribute**

Binding\_indication template [86](#)

**entity\_declaration attribute**

Architecture\_body template [75](#)

**entity\_designator attribute**

Attribute\_specification template [83](#)

**error\_id attribute**

Identifier template [133](#)

**evaluation\_time attribute**

Binary\_operation template [85](#)

Expression template [117](#)

For\_loop\_statement template [124](#)

Literal template [141](#)

Range template [159](#)

While\_loop\_statement template [190](#)

**exit\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

**expression attribute**

Allocator template [72](#)

Asynchronous\_initialization template [80](#)

Attribute\_name template [82](#)  
Attribute\_specification template [83](#)  
Case\_statement template [91](#)  
Clock template [93](#)  
Indexed\_name template [136](#)  
Return\_statement template [162](#)  
Selected\_signal\_assignment template [164](#)  
Synchronous\_initialization template [182](#)  
Variable\_assignment\_statement template [185](#)

## F

### **file attribute**

Attribute\_specification template [84](#)  
Formal\_parameter template [125](#)  
Interface\_file\_declaration template [138](#)

### **file\_declaration attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Entity\_declaration template [112](#)  
Generate\_statement template [128](#)  
Package\_body template [146](#)  
Package\_declaration template [149](#)  
Process\_statement template [156](#)  
Subprogram\_body template [174](#)

### **file\_layout attribute**

Architecture\_body template [75](#)  
Configuration\_declaration template [104](#)  
Entity\_declaration template [113](#)  
Package\_body template [147](#)

Package\_declaration template [149](#)

### **file\_length attribute**

Architecture\_body template [75](#)

Entity\_declaration template [113](#)

File\_layout template [120](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

### **file\_length\_count attribute**

Configuration\_declaration template [104](#)

### **file\_name attribute**

Architecture\_body template [75](#)

Configuration\_declaration template [104](#)

Entity\_declaration template [113](#)

File\_layout template [120](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

### **fixed\_value attribute**

Clock template [93](#)

Data signal template [58](#)

Data\_signal template [58](#)

### **flipflop attribute**

Design template [59](#)

Indexed\_name template [136](#)

Port\_declaration template [152](#)

Process\_statement template [157](#)

Signal\_declaration template [168](#)

Simple\_name template [170](#)

Slice\_name template [171](#)

**flipflop\_as\_source attribute**

Connectivity\_path template [56](#)

**for\_loop\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process statement template [156](#)

Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

**formal\_function attribute**

Association\_element template [78](#)

**formal\_parameter attribute**

Subprogram\_declaration template [178](#)

**formal\_type\_mark attribute**

Association\_element template [78](#)

**fsm attribute**

Architecture\_body template [75](#)

Process\_statement template [157](#)

**fully\_assign\_signals attribute**

Process\_statement template [157](#)

**fully\_assign\_variables attribute**

Process\_statement template [157](#)

Subprogram\_body template [174](#)

**function attribute**Attribute\_specification template [83](#)Subprogram\_declaration template [178](#)**function\_cost attribute**Logic\_cost template [63](#)**G****gated\_clock attribute**Design template [59](#)**gated\_in\_unit attribute**Asynchronous\_initialization template [80](#)Clock template [93](#)Synchronous\_initialization template [182](#)**gated\_reset attribute**Design template [59](#)**generate\_statement attribute**Architecture\_body template [74](#)Block\_statement template [90](#)Generate\_statement template [129](#)Generate\_statement template [129](#)**generate\_statement\_label attribute**Block\_specification template [88](#)**generic\_declaration attribute**Block\_statement template [89](#)Component\_declaration template [95](#)

Entity\_declaration template [112](#)

**generic\_map\_aspect attribute**

Binding\_indication template [86](#)

Block\_statement template [89](#)

Component\_instantiation\_statement template [97](#)

**glue\_logic\_at\_top attribute**

Design template [59](#)

**group attribute**

Attribute\_specification template [84](#)

**group\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Configuration\_declaration template [104](#)

Entity\_declaration template [112](#)

Generate\_statement template [129](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

**group\_template\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Entity\_declaration template [112](#)

Generate\_statement template [129](#)

Package\_body template [146](#)

Package\_declaration template [149](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

**guard\_expression attribute**

Block\_statement template [89](#)

**guarded attribute**

Conditional\_signal\_assignment template [102](#)

Selected\_signal\_assignment template [164](#)

## H

**has\_clock\_as\_data attribute**

Flipflop template [61](#), [122](#)

Latch template [62](#), [140](#)

**header\_comment attribute**

Architecture\_body template [75](#)

Configuration\_declaration template [104](#)

Entity\_declaration template [113](#)

File\_layout template [120](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

Process\_statement template [157](#)

Subprogram\_body template [175](#)

Subprogram\_declaration template [178](#)

**high\_bound attribute**

Range template [159](#)

**hold\_latch\_data attribute**

Test\_signal template [65](#)



# I

## **identifier attribute**

Alias\_declaration template [71](#)  
Architecture\_body template [73](#)  
Asynchronous\_initialization template [80](#)  
Attribute\_declaration template [81](#)  
Clock template [93](#)  
Component\_declaration template [95](#)  
Concurrent\_procedure\_call\_statement template [100](#)  
Configuration\_declaration template [104](#)  
Constant\_declaration template [106](#)  
Element\_declaration template [110](#)  
Entity\_declaration template [112](#)  
Enumeration\_type\_definition template [115](#)  
File\_declaration template [119](#)  
File\_layout template [120](#)  
Flipflop template [60](#), [121](#)  
Formal\_parameter template [125](#)  
Function\_call template [126](#)  
Generic\_declaration template [130](#)  
Indexed\_name template [136](#)  
Interface\_file\_declaration template [138](#)  
Interface\_variable\_declaration template [139](#)  
Latch template [62](#), [140](#)  
Package\_declaration template [148](#)  
Physical\_type\_definition template [150](#)  
Port\_declaration template [151](#)  
Procedure\_call\_statement template [154](#)  
Selected\_name template [163](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [168](#)

Simple\_name template [170](#)  
Slice\_name template [171](#)  
Subprogram\_declaration template [178](#)  
Subtype\_declaration template [180](#)  
Synchronous\_initialization template [182](#)  
Type\_declaration template [183](#)  
Variable\_declaration template [186](#)

### **if\_statement attribute**

Case\_statement template [91](#)  
For\_loop\_statement template [123](#)  
If\_statement template [134](#)  
Loop\_statement template [143](#)  
Process\_statement template [156](#)  
Subprogram\_body template [174](#)  
While\_loop\_statement template [190](#)

### **impure attribute**

Subprogram\_declaration template [178](#)

### **in attribute**

Formal\_parameter template [125](#)  
Generic\_declaration template [130](#)  
Interface\_variable\_declaration template [139](#)  
Port\_declaration template [151](#)

### **incomplete\_type\_declaration attribute**

Type\_declaration template [183](#)

### **indentation\_depth attribute**

If\_statement template [134](#)  
Statement\_format template [172](#)

**index\_constraint\_s attribute**

Constrained\_array\_definition template [107](#)

**index\_specification attribute**

Block\_specification template [88](#)

**inertial attribute**

Conditional\_signal\_assignment template [102](#)

Selected\_signal\_assignment template [164](#)

Signal\_assignment\_statement template [167](#)

**initialization\_count attribute**

Design template [59](#)

**initialize\_signals attribute**

Process\_statement template [157](#)

**initialize\_variables attribute**

Process\_statement template [157](#)

**inout attribute**

Formal\_parameter template [125](#)

Interface\_variable\_declaration template [139](#)

Port\_declaration template [151](#)

**input attribute**

Flipflop template [60](#), [121](#)

Latch template [62](#), [140](#)

**inverter\_count attribute**

Connectivity\_path template [56](#)

**is\_clock attribute**

Port\_declaration template [151](#)

Signal\_declaration template [169](#)

**is\_combinatorial attribute**

Connectivity\_path template [56](#)

**is\_load attribute**

Asynchronous\_initialization template [80](#)

Synchronous\_initialization template [182](#)

**is\_reset attribute**

Asynchronous\_initialization template [80](#)

Connectivity\_path template [56](#)

Port\_declaration template [151](#)

Signal\_declaration template [169](#)

Synchronous\_initialization template [182](#)

**is\_set attribute**

Asynchronous\_initialization template [80](#)

Synchronous\_initialization template [182](#)

**L****label attribute**

Assertion statement template [77](#)

Assertion\_statement template [77](#)

Attribute\_specification template [83](#)

Block\_statement template [89](#)

Case\_statement template [91](#)

Component\_instantiation\_statement template [97](#)

Component\_specification template [98](#)

Concurrent\_assertion\_statement template [99](#)  
Concurrent\_procedure\_call\_statement template [100](#)  
Conditional\_signal\_assignment template [101](#)  
Exit\_statement template [116](#)  
For\_loop\_statement template [123](#)  
Generate\_statement template [128](#)  
If\_statement template [134](#)  
Loop\_statement template [143](#)  
Next\_statement template [145](#)  
Procedure\_call\_statement template [154](#)  
Process\_statement template [155](#)  
Report\_statement template [161](#)  
Return\_statement template [162](#)  
Selected\_signal\_assignment template [164](#)  
Signal\_assignment\_statement template [167](#)  
Variable\_assignment\_statement template [185](#)  
Wait\_statement template [187](#)  
While\_loop\_statement template [189](#)

### **latch attribute**

Design template [59](#)  
Indexed\_name template [136](#)  
Port\_declaration template [152](#)  
Process\_statement template [157](#)  
Signal\_declaration template [168](#)  
Simple\_name template [170](#)  
Slice\_name template [171](#)

### **latch\_as\_source attribute**

Connectivity\_path template [56](#)

**left\_expression attribute**Binary\_operation template [85](#)Range template [159](#)**library\_clause attribute**Design\_unit template [108](#)**library\_name attribute**Design\_unit template [108](#)**limit\_id attribute**Identifier template [133](#)**line\_count attribute**Architecture\_body template [75](#)Configuration\_declaration template [104](#)Entity\_declaration template [113](#)Package\_body template [147](#)Package\_declaration template [149](#)Statement\_format template [172](#)**linkage attribute**Interface\_variable\_declaration template [139](#)Port\_declaration template [151](#)**literal attribute**Attribute\_specification template [84](#)**load\_count attribute**Design template [59](#)

**logic\_level attribute**

Design template [59](#)

**loop\_label attribute**

Exit\_statement template [116](#)

Next\_statement template [145](#)

**loop\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

**low\_bound attribute**

Range template [159](#)

## M

**max\_cost attribute**

Logic\_cost template [63](#)

**maximum\_variable\_usage attribute**

Process\_statement template [157](#)

**mealy attribute**

Fsm template [118](#)

**meta\_stability attribute**

Design template [59](#)

**minus\_cost attribute**

Logic\_cost template [63](#)

**missing\_signals\_in\_sensitivity\_list attribute**

Process\_statement template [157](#)

**mixed\_aync\_sync\_line attribute**

Design template [59](#)

**mixed\_clock attribute**

Design template [59](#)

**modulus\_cost attribute**

Logic\_cost template [63](#)

**moore attribute**

Fsm template [118](#)

**multiple\_choices attribute**

Aggregate template [70](#)

Case\_statement template [92](#)

Element\_association template [110](#)

**multiplexed\_clock attribute**

Design template [59](#)

**multiply\_cost attribute**

Logic\_cost template [63](#)

**mux\_cost attribute**

Logic\_cost template [63](#)



## N

### **name attribute**

Alias\_declaration template [71](#)

### **name\_prefix attribute**

Attribute\_name template [82](#)

### **named\_association attribute**

Aggregate template [70](#)

Association\_element template [78](#)

Association\_list template [79](#)

Element\_association template [110](#)

### **named\_entity\_prefix attribute**

Attribute\_name template [82](#)

### **nand\_cost attribute**

Logic\_cost template [63](#)

### **ncs\_file\_name attribute**

Architecture\_body template [75](#)

Configuration\_declaration template [104](#)

Entity\_declaration template [113](#)

Package\_body template [147](#)

Package\_declaration template [149](#)

### **next\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)  
Subprogram\_body template [175](#)  
While\_loop\_statement template [190](#)

**non\_tristate\_drivers\_per\_signal attribute**  
design template [59](#)

**nor\_cost attribute**  
Logic\_cost template [63](#)

**null attribute**  
Literal template [141](#)  
Waveform\_element template [188](#)

**null\_range attribute**  
Range template [159](#)

**null\_statement attribute**  
Case\_statement template [92](#)  
For\_loop\_statement template [124](#)  
If\_statement template [135](#)  
Loop\_statement template [144](#)  
Process\_statement template [156](#)  
Subprogram\_body template [175](#)  
While\_loop\_statement template [190](#)

## O

**object\_definition attribute**  
Asynchronous\_initialization template [80](#)  
Clock template [93](#)  
Flipflop template [60](#), [122](#)

Indexed\_name template [136](#)  
Latch template [62](#)  
Selected\_name template [163](#)  
Simple\_name template [170](#)  
Slice\_name template [171](#)  
Synchronous\_initialization template [182](#)

### **one\_declaration\_per\_line attribute**

Alias\_declaration template [71](#)  
Attribute\_declaration template [81](#)  
Constant\_declaration template [107](#)  
File\_declaration template [119](#)  
Formal\_parameter template [125](#)  
Generic\_declaration template [130](#)  
Interface\_file\_declaration template [138](#)  
Interface\_variable\_declaration template [139](#)  
Port\_declaration template [152](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [169](#)  
Variable\_declaration template [186](#)

### **one\_statement\_per\_line attribute**

Statement\_format template [172](#)

### **open attribute**

Association\_element template [78](#)  
Association\_list template [79](#)

### **open\_kind attribute**

File\_declaration template [119](#)

**operand attribute**

Expression template [117](#)

**operand\_size\_match attribute**

Binary\_operation template [85](#)

**operand\_size\_mismatch attribute**

Binary\_operation template [85](#)

Conditional\_signal\_assignment template [101](#)

Signal\_assignment\_statement template [167](#)

Variable\_assignment\_statement template [185](#)

**operator\_symbol attribute**

Alias\_declaration template [71](#)

Binary\_operation template [85](#)

Expression template [117](#)

Subprogram\_declaration template [178](#)

**or\_cost attribute**

Logic\_cost template [63](#)

**others attribute**

Aggregate template [71](#)

Association\_element template [78](#)

Association\_list template [79](#)

Attribute\_specification template [83](#)

Case\_statement template [92](#)

Component\_specification template [99](#)

Configuration\_specification template [106](#)

Disconnection\_specification template [109](#)

Element\_association template [110](#)

Selected\_waveforms template [165](#)

**out attribute**

Formal\_parameter template [125](#)

Interface\_variable\_declaration template [139](#)

Port\_declaration template [151](#)

**out\_params\_fully\_assigned attribute**

Subprogram\_body template [175](#)

**output attribute**

Flipflop template [60](#), [121](#), [122](#)

Latch template [62](#), [140](#)

**outputs\_driven attribute**

Port\_declaration template [152](#)

Signal\_declaration template [168](#)

**P****package attribute**

Attribute\_specification template [83](#)

**parameter\_count attribute**

Association\_list template [79](#)

Subprogram\_declaration template [178](#)

**plus\_cost attribute**

Logic\_cost template [63](#)

**port\_count attribute**

Component\_declaration template [96](#)

Entity\_declaration template [113](#)

**port\_declaration attribute**

Block\_statement template [89](#)  
Component\_declaration template [95](#)  
Entity\_declaration template [112](#)

**port\_map\_aspect attribute**

Binding\_indication template [86](#)  
Block\_statement template [89](#)  
Component\_instantiation\_statement template [97](#)

**port\_order attribute**

Component\_declaration template [96](#)  
Entity\_declaration template [113](#)

**positional\_association attribute**

Aggregate template [70](#)  
Association\_element template [78](#)  
Association\_list template [79](#)  
Element\_association template [109](#)

**postponed attribute**

Assertion\_statement template [77](#)  
Component\_specification template [99](#)  
Concurrent\_assertion\_statement template [100](#)  
Concurrent\_procedure\_call\_statement template [100](#)  
Conditional\_signal\_assignment template [101](#)  
Process\_statement template [156](#)  
Selected\_signal\_assignment template [164](#)

**power\_cost attribute**

Logic\_cost template [63](#)

**procedure attribute**

Attribute\_specification template [83](#)

Subprogram\_declaration template [178](#)

**procedure\_call\_statement attribute**

Block\_statement template [90](#)

Case\_statement template [91](#)

Entity\_declaration template [112](#)

For\_loop\_statement template [123](#)

Generate\_statement template [129](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

While\_loop\_statement template [190](#)

**process\_statement attribute**

Architecture\_body template [74](#)

Block\_statement template [90](#)

Entity\_declaration template [112](#)

Generate\_statement template [129](#)

**pulse\_generator attribute**

design template [59](#)

**pure attribute**

Subprogram\_declaration template [178](#)

**Q****qualified attribute**

Expression template [117](#)

## R

### **range attribute**

Floating\_type\_definition template [122](#)

Integer\_type\_definition template [137](#)

Physical\_type\_definition template [150](#)

Slice\_name template [171](#)

Subtype\_indication template [181](#)

### **range\_attribute attribute**

Range template [159](#)

### **range\_index attribute**

Range template [159](#)

### **range\_overflow attribute**

Association\_element template [78](#)

Conditional\_signal\_assignment template [102](#)

Element\_association template [109](#)

Indexed\_name template [136](#)

Selected\_signal\_assignment template [164](#)

Signal\_assignment\_statement template [167](#)

Slice\_name template [171](#)

Variable\_assignment\_statement template [185](#)

### **reach\_memory attribute**

Test\_signal template [65](#)

### **read\_write attribute**

Association\_list template [79](#)

Conditional\_signal\_assignment template [102](#)

Selected\_signal\_assignment template [164](#)



Signal\_assignment\_statement template [167](#)

Variable\_assignment\_statement template [185](#)

### **record\_aggregate attribute**

Aggregate template [70](#)

### **recursion\_type attribute**

Subprogram\_body template [175](#)

### **redundancy\_in\_sensitivity\_list attribute**

Process\_statement template [157](#)

### **register attribute**

Port\_declaration template [152](#)

Signal\_declaration template [168](#)

### **registered\_inputs attribute**

Design template [59](#)

### **registered\_outputs attribute**

Design template [59](#)

### **reject\_expression attribute**

Conditional\_signal\_assignment template [101](#)

Selected\_signal\_assignment template [164](#)

Signal\_assignment\_statement template [167](#)

### **related\_unit attribute**

Architecture\_body template [74](#)

Configuration\_declaration template [104](#)

### **related\_units attribute**

File\_layout template [120](#)

**remainder\_cost attribute**

Logic\_cost template [63](#)

**report\_expression attribute**

Assertion\_statement template [77](#)

Concurrent\_assertion\_statement template [99](#)

Report\_statement template [161](#)

**report\_statement attribute**

Case\_statement template [91](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

While\_loop\_statement template [190](#)

**reset\_at\_hierarchical\_boundary attribute**

Logic\_cost template [63](#)

**reset\_count attribute**

Design template [59](#)

**resolution\_function attribute**

Subtype\_indication template [181](#)

**return\_last attribute**

Subprogram\_body template [175](#)

**return\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)  
Loop\_statement template [143](#)  
Process\_statement template [156](#)  
Subprogram\_body template [175](#)  
While\_loop\_statement template [190](#)

### **right\_expression attribute**

Binary\_operation template [85](#)  
Range template [159](#)

## **S**

### **selected\_name attribute**

Use\_clause template [185](#)

### **selected\_signal\_assignment attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Generate\_statement template [129](#)

### **selected\_waveform\_s attribute**

Selected\_signal\_assignment template [164](#)

### **sensitivity attribute**

Process\_statement template [155](#)  
Wait\_statement template [187](#)

### **sensitivity\_count attribute**

Process\_statement template [157](#)  
Wait\_statement template [187](#)

**set\_count attribute**

Design template [59](#)

**severity\_expression attribute**

Assertion\_statement template [77](#)

Concurrent\_assertion\_statement template [100](#)

Report\_statement template [161](#)

**shared\_variable\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [89](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

**shift\_cost attribute**

Logic\_cost template [63](#)

**signal attribute**

Attribute\_specification template [84](#)

Formal\_parameter template [125](#)

Port\_declaration template [151](#)

**signal\_assignment\_statement attribute**

Case\_statement template [91](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

While\_loop\_statement template [190](#)

**signal\_declaration attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Entity\_declaration template [112](#)  
Generate\_statement template [128](#)  
Package\_declaration template [149](#)

**signals\_driven attribute**

Port\_declaration template [151](#)  
Signal\_declaration template [168](#)

**starting\_unit attribute**

Clock template [93](#)  
Connectivity\_path template [56](#)

**state\_count attribute**

Fsm template [118](#)

**state\_variable attribute**

Fsm template [118](#)

**statement\_format attribute**

Architecture\_body template [74](#)  
Block\_statement template [90](#)  
Case\_statement template [92](#)  
Conditional\_signal\_assignment template [102](#)  
For\_loop\_statement template [124](#)  
If\_statement template [135](#)  
Loop\_statement template [144](#)  
Package\_body template [147](#)  
Package\_declaration template [149](#)  
Process\_statement template [157](#)

Selected\_signal\_assignment template [164](#)

Subprogram\_body template [175](#)

### **statement\_profile\_s attribute**

Architecture\_body template [75](#)

Block\_statement template [90](#)

Case\_statement template [92](#)

For\_loop\_statement template [124](#)

If\_statement template [135](#)

Loop\_statement template [144](#)

Process\_statement template [157](#)

Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

### **subprogram\_body attribute**

Architecture\_body template [73](#)

Block\_statement template [89](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

Process\_statement template [155](#)

Subprogram\_body template [174](#)

Subprogram\_declaration template [178](#)

### **subprogram\_declaration attribute**

Architecture\_body template [73](#)

Block\_statement template [89](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

Package\_declaration template [148](#)

Process\_statement template [155](#)

Subprogram\_body template [174](#)

### **subtype attribute**

Attribute\_specification template [84](#)

### **subtype\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [89](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

Package\_declaration template [148](#)

Process\_statement template [155](#)

Subprogram\_body template [174](#)

### **subtype\_definition attribute**

Unconstrained\_array\_definition template [184](#)

### **subtype\_indication attribute**

Access\_type\_definition template [70](#)

Access\_type\_definition template [70](#)

Alias\_declaration template [71](#)

Allocator template [72](#)

Constant\_declaration template [106](#)

Constrained\_array\_definition template [107](#)

Element\_declaration template [110](#)

File\_declaration template [119](#)

For\_loop\_statement template [123](#)

Formal\_parameter template [125](#)

Generic\_declaration template [130](#)

Interface\_file\_declaration template [138](#)

Interface\_variable\_declaration template [139](#)

Port\_declaration template [151](#)  
Shared\_variable\_declaration template [166](#)  
Signal\_declaration template [168](#)  
Subtype\_declaration template [180](#)  
Subtype\_indication template [181](#)  
Unconstrained\_array\_definition template [184](#)  
Variable\_declaration template [186](#)

### **sync\_ff\_count attribute**

Design template [59](#)

### **synchronous\_initialization attribute**

Design template [58](#)

Latch template [62](#), [140](#)

### **synchronous\_reset attribute**

Process\_statement template [157](#)

### **synchronous\_reset\_signal attribute**

Architecture\_body template [75](#)

## **T**

### **target attribute**

Conditional\_signal\_assignment template [101](#)

Selected\_signal\_assignment template [164](#)

Signal\_assignment\_statement template [167](#)

Variable\_assignment\_statement template [185](#)

### **text attribute**

Comment template [94](#)



**timeout attribute**

Wait\_statement template [187](#)

**to attribute**

Range template [159](#)

**top\_architecture attribute**

Architecture\_body template [75](#)

**top\_configuration attribute**

Configuration\_declaration template [104](#)

**top\_entity attribute**

Entity\_declaration template [113](#)

**top\_unit attribute**

Design template [58](#)

**transition\_in\_default attribute**

Fsm template [118](#)

**transport attribute**

Conditional\_signal\_assignment template [102](#)

Selected\_signal\_assignment template [164](#)

Signal\_assignment\_statement template [167](#)

**tristate attribute**

Port\_declaration template [152](#)

Signal\_declaration template [169](#)

**type attribute**

Attribute\_specification template [84](#)

**type\_conversion attribute**

Expression template [117](#)

**type\_declaration attribute**

Architecture\_body template [74](#)

Block\_statement template [89](#)

Entity\_declaration template [112](#)

Generate\_statement template [128](#)

Package\_body template [146](#)

Package\_declaration template [148](#)

Process\_statement template [155](#)

Subprogram\_body template [174](#)

**type\_definition attribute**

Type\_declaration template [183](#)

**type\_mark attribute**

Attribute\_declaration template [81](#)

Binary\_operation template [85](#)

Disconnection\_specification template [109](#)

File\_type\_definition template [121](#)

Record\_type\_definition template [161](#)

Subprogram\_declaration template [178](#)

Subtype\_indication template [181](#)

**U****unaffected attribute**

Waveform template [189](#)

**unit\_count attribute**

File\_layout template [120](#)

**unit\_name attribute**

Component\_instantiation\_statement template [97](#)

**units attribute**

Attribute\_specification template [84](#)

**unused\_declaration attribute**

Architecture\_body template [75](#)

Process\_statement template [156](#)

**use\_clause attribute**

Architecture\_body template [74](#)

Block\_configuration template [87](#)

Block\_statement template [90](#)

Configuration\_declaration template [104](#)

Design\_unit template [108](#)

Entity\_declaration template [112](#)

Generate\_statement template [129](#)

Package\_body template [146](#)

Package\_declaration template [149](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

**use\_db\_name attribute**

Component\_instantiation\_statement template [97](#)

Entity\_declaration template [112](#)

**use\_exponent attribute**

Literal template [141](#)

**use\_work attribute**

Design\_unit template [108](#)

## V

### **value attribute**

Binary\_operation template [85](#)

Expression template [117](#)

Literal template [141](#)

### **value\_type attribute**

Literal template [141](#)

### **variable attribute**

Attribute\_specification template [84](#)

Formal\_parameter template [125](#)

### **variable\_assignment\_statement attribute**

Case\_statement template [91](#)

For\_loop\_statement template [123](#)

If\_statement template [134](#)

Loop\_statement template [143](#)

Process\_statement template [156](#)

Subprogram\_body template [174](#)

While\_loop\_statement template [190](#)

### **variable\_declaration attribute**

Process\_statement template [156](#)

Subprogram\_body template [174](#)

## W

### **wait\_statement attribute**

Case\_statement template [92](#)

For\_loop\_statement template [124](#)

If\_statement template [134](#)  
Loop\_statement template [143](#)  
Process\_statement template [156](#)  
Subprogram\_body template [175](#)  
While\_loop\_statement template [190](#)

### **waveform attribute**

Conditional\_waveform template [103](#)  
Selected\_waveforms template [165](#)  
Signal\_assignment\_statement template [167](#)

### **waveform\_count attribute**

Conditional\_signal\_assignment template [102](#)  
Selected\_signal\_assignment template [164](#)  
Signal\_assignment\_statement template [167](#)

### **waveform\_element\_count attribute**

Waveform template [189](#)

### **waveform\_element\_s attribute**

Waveform template [189](#)

### **waveform\_expression attribute**

Waveform\_element template [188](#)

### **while\_loop\_statement attribute**

Case\_statement template [92](#)  
For\_loop\_statement template [123](#)  
If\_statement template [134](#)  
Loop\_statement template [143](#)  
Process\_statement template [156](#)  
Subprogram\_body template [175](#)

While\_loop\_statement template [190](#)

**within\_same\_clkdomain attribute**

Connectivity\_path template [56](#)

## **X**

**xnor\_cost attribute**

Logic\_cost template [63](#)

**xor\_cost attribute**

Logic\_cost template [63](#)

# Index

## A

Access\_type\_definition template 70  
 Aggregate template 70  
 aggregate\_attributes 26  
 Alias\_declaration template 71  
 Allocator template 72  
 Architecture\_body template 73  
 Assertion\_statement template 77  
 Association\_element template 78  
 Association\_list template 79  
 Asynchronous\_initialization template 80  
 Attribute\_declaration template 81  
 Attribute\_name template 82  
 Attribute\_specification template 83  
 Attributes  
   aggregate 19, 26, 27

## B

Binary\_operation template 85  
 Binding\_indication template 86  
 Block\_configuration template 87  
 Block\_specification template 88  
 block\_statement 51  
 Block\_statement template 89

## C

C 15  
 Case\_statement template 91  
 Class  
   CONCURRENT\_STATEMENT 48  
   DECLARATIVE\_ITEM 48  
   DISCRETE\_RANGE 49  
   DRIVEN\_OBJECT 49  
   EXPRESSION 49  
   NAME 50  
   OBJECT\_ITEM 51  
   REGION\_PART 51  
   SEQUENTIAL\_STATEMENT 52

STATEMENT 53  
 TARGET 54  
 TYPE\_DEFINITION 54  
 TYPE\_MARK 54  
 Clock template 93  
 Commands  
   VRSL 16  
 Comment template 94  
 Component\_configuration template 95  
 component\_declaration 51  
 Component\_declaration template 95  
 Component\_instantiation\_statement  
   template 97  
 Component\_specification template 98  
 Concurrent\_assertion\_statement template  
   99  
 Concurrent\_procedure\_call\_statement  
   template 100  
 CONCURRENT\_STATEMENT Class 48  
 Conditional\_signal\_assignment template  
   101  
 Conditional\_waveforms template 103  
 Configuration\_declaration template 104  
 Configuration\_specification template 106  
 Connectivity\_path template 56  
 Constant\_declaration template 106  
 Constrained\_array\_definition template  
   107

## D

Data\_signal template 58  
 DECLARATIVE\_ITEM Class 48  
 Design template 58  
 Design\_unit template 108  
 Disconnection\_specification template 109  
 DISCRETE\_RANGE Class 49  
 DRIVEN\_OBJECT Class 49

**E**

Element\_association template [109](#)  
 Element\_declaration template [110](#)  
 Entity\_declaration template [112](#)  
 Enumeration\_type template [115](#)  
 Environment variables  
   LEDA\_HTML\_DOC\_PATH [31](#)  
   LEDA\_HTML\_USR\_PATH [31](#)  
 Error message parameters  
   <%actual> [33](#)  
   <%context> [32](#)  
   <%formal> [33](#)  
   <%item> [32](#)  
   <%value> [33](#)  
 Error messages, parameterizing [32](#)  
 exit\_statement [52](#)  
 Exit\_statement template [116](#)  
 EXPRESSION Class [49](#)  
 Expression template [117](#)

**F**

File\_declaration template [119](#)  
 File\_layout template [120](#)  
 File\_type\_definition template [121](#)  
 Files  
   HTML [31](#)  
 Flipflop template [60](#), [121](#)  
 Floating\_type\_definition template [122](#)  
 for\_loop\_statement [52](#)  
 For\_loop\_statement template [123](#)  
 Formal\_parameter template [125](#)  
 Fsm template [118](#)  
 Function\_call template [126](#)

**G**

generate\_statement [52](#)  
 Generate\_statement template [128](#)  
 generic\_declaration [51](#)  
 Generic\_declaration template [130](#)  
 Group declarations [131](#)  
 Group template declarations [131](#)

**H**

Hardware rules [15](#)  
 Header\_comment template [132](#)  
 HTML files  
   linking to [31](#)

**I**

Identifier template [133](#)  
 If\_statement template [134](#)  
 Indexed\_name template [136](#)  
 Integer\_type\_definition template [137](#)  
 Integrating C-based rules [15](#)  
 Integrating Tcl-based rules [15](#)  
 Interface Variable Declaration Template  
   [139](#)  
 Interface\_file\_declaration template [138](#)  
 Interface\_variable\_declaration template  
   [139](#)

**L**

Latch template [62](#), [140](#)  
 Leda C Interface Guide [15](#)  
 Leda Tcl Interface Guide [15](#)  
 Literal template [141](#)  
 Logic\_cost template [63](#)  
 loop\_statement [52](#)  
 Loop\_statement template [143](#)

**M**

Manuals  
   LRM [16](#)  
   VHDL Language Reference [16](#)  
 Max/min attributes  
   constraining [30](#)

**N**

NAME Class [50](#)  
 next\_statement [52](#)  
 Next\_statement template [145](#)



**O**

OBJECT\_ITEM Class 51

**P**

Package\_body template 146  
 Package\_declaration template 148  
 Parameters  
   for error messages 32  
   predefined 29  
 Physical\_type\_definition template 150  
 port\_declaration 49, 51  
 Port\_declaration template 151  
 Procedure\_call\_statement template 154  
 process\_statement 52  
 Process\_statement template 155

**R**

Range 43  
 Range template 159  
 Record\_type\_definition template 161  
 REGION\_PART Class 51  
 Regular expressions 133  
 Related documents 11  
 return\_statement 52  
 Return\_statement template 162  
 Ruleset 17, 36

**S**

Selected\_name template 163  
 Selected\_signal\_assignment template 164  
 Selected\_waveforms template 165  
 SEQUENTIAL\_STATEMENT Class 52  
 Shared Variable Declaration Template 166  
 shared\_variable\_declaration 166  
 Shared\_variable\_declaration template 166  
 Signal\_assignment template 167  
 Signal\_declaration template 168  
 Simple\_name template 170  
 Slice\_name template 171  
 STATEMENT Class 53  
 Statement\_format template 172

Subprogram\_body template 174  
 Subprogram\_declaration template 178  
 Subtype\_declaration template 180  
 Subtype\_indication template 181  
 Synchronous\_initialization template 182

**T**

TARGET Class 54  
 Tcl 15  
 Tcl-based rules 15  
 TEMPLATES  
   Access\_type\_definition 70  
   Aggregate 70  
   Alias\_declaration 71  
   Allocator 72  
   Architecture\_body 73  
   Assertion\_statement 77  
   Association\_element 78  
   Association\_list 79  
   Asynchronous\_initialization 80  
   Attribute\_declaration 81  
   Attribute\_name 82  
   Attribute\_specification 83  
   Binary\_operation 85  
   Binding\_indication 86  
   Block\_configuration 87  
   Block\_specification 88  
   Block\_statement 89  
   Case\_statement 91  
   Clock 93  
   Comment 94  
   Component\_configuration 95  
   Component\_declaration 95  
   Component\_instantiation\_statement 97  
   Component\_specification 98  
   Concurrent\_assertion\_statement 99  
   Concurrent\_procedure\_call\_statement 100  
   Conditional\_signal\_assignment 101  
   Conditional\_waveforms 103  
   Configuration\_declaration 104  
   Configuration\_specification 106  
   Connectivity\_path 56  
   Constant\_declaration 106

- Constrained\_array\_definition 107
  - Data\_signal 58
  - Design 58
  - Design\_unit 108
  - Disconnection\_specification 109
  - Element\_association 109
  - Element\_declaration 110
  - Entity\_declaration 112
  - Enumeration\_type 115
  - Exit\_template 116
  - Expression 117
  - File\_declaration 119
  - File\_layout 120
  - File\_type\_definition 121
  - Flipflop 60, 121
  - Floating\_type\_definition 122
  - For\_loop\_statement 123
  - Formal\_parameter 125
  - Fsm 118
  - Function\_call 126
  - Generate\_statement 128
  - Generic\_declaration 130
  - Header\_comment 132
  - Identifier 133
  - If\_statement 134
  - Indexed\_name 136
  - Integer\_type\_definition 137
  - Interface\_file\_declaration 138
  - Interface\_variable\_declaration 139
  - Latch 62, 140
  - Literal 141
  - Logic\_cost 63
  - Loop\_statement 143
  - Next\_statement 145
  - Package\_body 146
  - Package\_declaration 148
  - Physical\_type\_definition 150
  - Port\_declaration 151
  - Procedure\_call\_statement 154
  - Process\_statement 155
  - Range 159
  - Record\_type\_definition 161
  - Return\_statement 162
  - Selected\_name 163
  - Selected\_signal\_assignment 164
  - Selected\_waveforms 165
  - Shared\_variable\_declaration 166
  - Signal\_assignment 167
  - Signal\_declaration 168
  - Simple\_name 170
  - Slice\_name 171
  - Statement\_format 172
  - Subprogram\_body 174
  - Subprogram\_declaration 178
  - Subtype\_declaration 180
  - Subtype\_indication 181
  - Synchronous\_initialization 182
  - Test\_signal 65
  - Type\_declaration 183
  - Unconstrained\_array\_definition 184
  - Use\_clause 185
  - Variable\_assignment\_statement 185
  - Variable\_declaration 186
  - Wait\_statement 187
  - Waveform 189
  - Waveform\_element 188
  - While\_loop\_statement 189
  - Templateset 17, 27, 36
  - Test\_signal template 65
  - Type\_declaration template 183
  - TYPE\_DEFINITION Class 54
  - TYPE\_MARK Class 54
- U**
- Unconstrained\_array\_definition template 184
  - UNIX
    - regular expressions 28
  - Use\_clause template 185
- V**
- Variable Declaration Template 186
  - Variable\_assignment\_statement template 185
  - Variable\_declaration template 186
  - VRSL 16, 36
    - aggregate\_attributes 26
    - attributes 18

- classes [47](#)
- commands [16, 18](#)
- conditional\_commands [24](#)
- context [21](#)
- examples [22](#)
- formal definition [36](#)
- max and min commands [40](#)
- message [21](#)
- primitives [44](#)
- rules [17](#)
- rulesets [17](#)
- severity [22](#)
- templates and attributes [16](#)
- templatesets [17](#)
- what is it? [16](#)

## W

- Wait\_statement template [187](#)
- Waveform template [189](#)
- Waveform\_element template [188](#)
- while\_loop\_statement [52](#)
- While\_loop\_statement template [189](#)

