

# GNATcheck Reference Manual

---

## *Predefined Rules*

The GNAT Pro Ada Compiler

GNAT Pro Version 6.3.0w

Configuration level: \$Revision: 150355 \$

Date: \$Date: 2009-10-06 17:35:14 +0400 (Tue, 06 Oct 2009) \$

AdaCore

---

Copyright © 2009, AdaCore

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being “GNU Free Documentation License”, with the Front-Cover Texts being “GNATcheck Reference Manual - Predefined Rules”, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

---

# About This Manual

The `gnatcheck` tool in GNAT Pro can be used to enforce coding conventions by analyzing Ada source programs with respect to a set of *rules* supplied at tool invocation. This manual describes the complete set of predefined rules that `gnatcheck` can take as input.

## What This Guide Contains

This guide contains a description of each predefined `gnatcheck` rule, organized into categories.

- [Chapter 1 \[Style-Related Rules\], page 3](#), presents the rules related to general programming style.
- [Chapter 2 \[Feature Usage Rules\], page 17](#), presents rules that enforce usage patterns for specific features.
- [Chapter 3 \[Metrics-Related Rules\], page 23](#), presents rules that enforce compliance with metric thresholds.
- [Chapter 4 \[SPARK Ada Rules\], page 25](#), presents rules that enforce adherence to the Ada subset handled by the SPARK tools.
- [Appendix A \[List of Rules\], page 29](#), gives an alphabetized list of all predefined rules, for ease of reference.

The name of each rule (the “rule identifier”) denotes the condition that is detected and flagged by `gnatcheck`. The rule identifier is used as a parameter of the ‘+R’ or ‘-R’ switch to `gnatcheck`.

## What You Should Know Before Reading This Guide

You should be familiar with the Ada language and with the usage of GNAT Pro in general and with the `gnatcheck` tool in particular; please refer to the *GNAT Pro User’s Guide*.



# 1 Style-Related Rules

The rules in this chapter may be used to enforce various feature usages consistent with good software engineering, for example as described in *Ada 95 Quality and Style*.

## 1.1 Tasking

The rules in this section may be used to enforce various feature usages related to concurrency.

### 1.1.1 Multiple\_Entries\_In\_Protected\_Definitions

Flag each protected definition (i.e., each protected object/type declaration) that defines more than one entry. Diagnostic messages are generated for all the entry declarations except the first one. An entry family is counted as one entry. Entries from the private part of the protected definition are also checked.

This rule has no parameters.

### 1.1.2 Volatile\_Objects\_Without\_Address\_Clauses

Flag each volatile object that does not have an address clause.

The following check is made: if the pragma `Volatile` is applied to a data object or to its type, then an address clause must be supplied for this object.

This rule does not check the components of data objects, array components that are volatile as a result of the pragma `Volatile_Components`, or objects that are volatile because they are atomic as a result of pragmas `Atomic` or `Atomic_Components`.

Only variable declarations, and not constant declarations, are checked.

This rule has no parameters.

## 1.2 Object Orientation

The rules in this section may be used to enforce various feature usages related to Object-Oriented Programming.

### 1.2.1 Deep\_Inheritance\_Hierarchies

Flags a tagged derived type declaration or an interface type declaration if its depth (in its inheritance hierarchy) exceeds the value specified by the 'N' rule parameter.

The inheritance depth of a tagged type or interface type is defined as 0 for a type with no parent and no progenitor, and otherwise as  $1 + \max$  of the depths of the immediate parent and immediate progenitors.

This rule does not flag private extension declarations. In the case of a private extension, the corresponding full declaration is checked.

This rule has the following (mandatory) parameter for the '+R' option:

- N* Integer not less than -1 specifying the maximal allowed depth of any inheritance hierarchy. If the rule parameter is set to -1, the rule flags all the declarations of tagged and interface types.

### 1.2.2 Direct\_Calls\_To\_Primitives

Flags any non-dispatching call to a dispatching primitive operation, except for the common idiom where a primitive subprogram for a tagged type directly calls the same primitive subprogram of the type's immediate ancestor.

This rule has no parameters.

### 1.2.3 Too\_Many\_Parents

Flags any type declaration, single task declaration or single protected declaration that has more than 'N' parents, 'N' is a parameter of the rule. A parent here is either a (sub)type denoted by the subtype mark from the parent\_subtype\_indication (in case of a derived type declaration), or any of the progenitors from the interface list, if any.

This rule has the following (mandatory) parameters for the '+R' option:

- N* Positive integer specifying the maximal allowed number of parents.

### 1.2.4 Visible\_Components

Flags all the type declarations located in the visible part of a library package or a library generic package that can declare a visible component. A type is considered as declaring a visible component if it contains a record definition by its own or as a part of a record extension. Type declaration is flagged even if it contains a record definition that defines no components.

Declarations located in private parts of local (generic) packages are not flagged. Declarations in private packages are not flagged.

This rule has no parameters.

## 1.3 Portability

The rules in this section may be used to enforce various feature usages that support program portability.

### 1.3.1 Forbidden\_Attributes

Flag each use of the specified attributes. The attributes to be detected are named in the rule's parameters.

This rule has the following parameters:

- For the ‘+R’ option

*Attribute\_Designator*

Adds the specified attribute to the set of attributes to be detected and sets the detection checks for all the specified attributes ON. If *Attribute\_Designator* does not denote any attribute defined in the Ada standard or in [section “Implementation Defined Attributes” in GNAT Reference Manual](#), it is treated as the name of unknown attribute.

GNAT      All the GNAT-specific attributes are detected; this sets the detection checks for all the specified attributes ON.

ALL      All attributes are detected; this sets the rule ON.

- For the ‘-R’ option

*Attribute\_Designator*

Removes the specified attribute from the set of attributes to be detected without affecting detection checks for other attributes. If *Attribute\_Designator* does not correspond to any attribute defined in the Ada standard or in [section “Implementation Defined Attributes” in GNAT Reference Manual](#), this option is treated as turning OFF detection of all unknown attributes.

GNAT      Turn OFF detection of all GNAT-specific attributes

ALL      Clear the list of the attributes to be detected and turn the rule OFF.

Parameters are not case sensitive. If *Attribute\_Designator* does not have the syntax of an Ada identifier and therefore can not be considered as a (part of an) attribute designator, a diagnostic message is generated and the corresponding parameter is ignored. (If an attribute allows a static expression to be a part of the attribute designator, this expression is ignored by this rule.)

When more than one parameter is given in the same rule option, the parameters must be separated by commas.

If more than one option for this rule is specified for the gnatcheck call, a new option overrides the previous one(s).

The ‘+R’ option with no parameters turns the rule ON, with the set of attributes to be detected defined by the previous rule options. (By default this set is empty, so if the only option specified for the rule is ‘+RForbidden\_Attributes’ (with no parameter), then the rule is enabled, but it does not detect anything). The ‘-R’ option with no parameter turns the rule OFF, but it does not affect the set of attributes to be detected.

### 1.3.2 Forbidden\_Pragmas

Flag each use of the specified pragmas. The pragmas to be detected are named in the rule's parameters.

This rule has the following parameters:

- For the '+R' option

*Pragma\_Name*

Adds the specified pragma to the set of pragmas to be checked and sets the checks for all the specified pragmas ON. *Pragma\_Name* is treated as a name of a pragma. If it does not correspond to any pragma name defined in the Ada standard or to the name of a GNAT-specific pragma defined in [section "Implementation Defined Pragmas" in GNAT Reference Manual](#), it is treated as the name of unknown pragma.

GNAT      All the GNAT-specific pragmas are detected; this sets the checks for all the specified pragmas ON.

ALL        All pragmas are detected; this sets the rule ON.

- For the '-R' option

*Pragma\_Name*

Removes the specified pragma from the set of pragmas to be checked without affecting checks for other pragmas. *Pragma\_Name* is treated as a name of a pragma. If it does not correspond to any pragma defined in the Ada standard or to any name defined in [section "Implementation Defined Pragmas" in GNAT Reference Manual](#), this option is treated as turning OFF detection of all unknown pragmas.

GNAT      Turn OFF detection of all GNAT-specific pragmas

ALL        Clear the list of the pragmas to be detected and turn the rule OFF.

Parameters are not case sensitive. If *Pragma\_Name* does not have the syntax of an Ada identifier and therefore can not be considered as a pragma name, a diagnostic message is generated and the corresponding parameter is ignored.

When more than one parameter is given in the same rule option, the parameters must be separated by a comma.

If more than one option for this rule is specified for the `gnatcheck` call, a new option overrides the previous one(s).

The '+R' option with no parameters turns the rule ON with the set of pragmas to be detected defined by the previous rule options. (By default this set is empty, so if the only option specified for the rule is '+RForbidden\_Pragmas' (with no



parameter), then the rule is enabled, but it does not detect anything). The ‘-R’ option with no parameter turns the rule OFF, but it does not affect the set of pragmas to be detected.

### 1.3.3 Implicit\_SMALL\_For\_Fixed\_Point\_Types

Flag each fixed point type declaration that lacks an explicit representation clause to define its ‘Small’ value. Since ‘Small’ can be defined only for ordinary fixed point types, decimal fixed point type declarations are not checked.

This rule has no parameters.

### 1.3.4 Predefined\_Numeric\_Types

Flag each explicit use of the name of any numeric type or subtype defined in package `Standard`.

The rationale for this rule is to detect when the program may depend on platform-specific characteristics of the implementation of the predefined numeric types. Note that this rule is over-pessimistic; for example, a program that uses `String` indexing likely needs a variable of type `Integer`. Another example is the flagging of predefined numeric types with explicit constraints:

```
subtype My_Integer is Integer range Left .. Right;
Vy_Var : My_Integer;
```

This rule detects only numeric types and subtypes defined in `Standard`. The use of numeric types and subtypes defined in other predefined packages (such as `System.Any_Priority` or `Ada.Text_IO.Count`) is not flagged

This rule has no parameters.

### 1.3.5 Separate\_Numeric\_Error\_Handlers

Flags each exception handler that contains a choice for the predefined `Constraint_Error` exception, but does not contain the choice for the predefined `Numeric_Error` exception, or that contains the choice for `Numeric_Error`, but does not contain the choice for `Constraint_Error`.

This rule has no parameters.

## 1.4 Program Structure

The rules in this section may be used to enforce feature usages related to program structure.

### 1.4.1 Deeply\_Nested\_Generics

Flags a generic declaration nested in another generic declaration if the nesting level of the inner generic exceeds a value specified by the ‘N’ rule parameter.

The nesting level is the number of generic declarations that enclose the given (generic) declaration. Formal packages are not flagged by this rule.

This rule has the following (mandatory) parameters for the ‘+R’ option:

*N*            Positive integer specifying the maximal allowed nesting level for a generic declaration.

#### 1.4.2 Local\_Packages

Flag all local packages declared in package and generic package specs. Local packages in bodies are not flagged.

This rule has no parameters.

#### 1.4.3 Non\_Visible\_Exceptions

Flag constructs leading to the possibility of propagating an exception out of the scope in which the exception is declared. Two cases are detected:

- An exception declaration in a subprogram body, task body or block statement is flagged if the body or statement does not contain a handler for that exception or a handler with an `others` choice.
- A `raise` statement in an exception handler of a subprogram body, task body or block statement is flagged if it (re)raises a locally declared exception. This may occur under the following circumstances:
  - it explicitly raises a locally declared exception, or
  - it does not specify an exception name (i.e., it is simply `raise;`) and the enclosing handler contains a locally declared exception in its exception choices.

Renamings of local exceptions are not flagged.

This rule has no parameters.

#### 1.4.4 Raising\_External\_Exceptions

Flag any `raise` statement, in a program unit declared in a library package or in a generic library package, for an exception that is neither a predefined exception nor an exception that is also declared (or renamed) in the visible part of the package.

This rule has no parameters.

### 1.5 Programming Practice

The rules in this section may be used to enforce feature usages that relate to program maintainability.

### 1.5.1 Anonymous\_Arrays

Flag all anonymous array type definitions (by Ada semantics these can only occur in object declarations).

This rule has no parameters.

### 1.5.2 Enumeration\_Ranges\_In\_CASE\_Statements

Flag each use of a range of enumeration literals as a choice in a `case` statement. All forms for specifying a range (explicit ranges such as `A .. B`, subtype marks and `'Range` attributes) are flagged. An enumeration range is flagged even if contains exactly one enumeration value or no values at all. A type derived from an enumeration type is considered as an enumeration type.

This rule helps prevent maintenance problems arising from adding an enumeration value to a type and having it implicitly handled by an existing `case` statement with an enumeration range that includes the new literal.

This rule has no parameters.

### 1.5.3 Exceptions\_As\_Control\_Flow

Flag each place where an exception is explicitly raised and handled in the same subprogram body. A `raise` statement in an exception handler, package body, task body or entry body is not flagged.

The rule has no parameters.

### 1.5.4 Exits\_From\_Conditional\_Loops

Flag any exit statement if it transfers the control out of a `for` loop or a `while` loop. This includes cases when the exit statement applies to a `FOR` or `while` loop, and cases when it is enclosed in some `for` or `while` loop, but transfers the control from some outer (unconditional) loop statement.

The rule has no parameters.

### 1.5.5 EXIT\_Statements\_With\_No\_Loop\_Name

Flag each `exit` statement that does not specify the name of the loop being exited.

The rule has no parameters.

### 1.5.6 GOTO\_Statements

Flag each occurrence of a `goto` statement.

This rule has no parameters.

### 1.5.7 Improper\_Returns

Flag each explicit `return` statement in procedures, and multiple `return` statements in functions. Diagnostic messages are generated for all `return` statements in a procedure (thus each procedure must be written so that it returns implicitly at the end of its statement part), and for all `return` statements in a function after the first one. This rule supports the stylistic convention that each subprogram should have no more than one point of normal return.

This rule has no parameters.

### 1.5.8 Non\_Short\_Circuit\_Operators

Flag all calls to predefined `and` and `or` operators for any boolean type. Calls to user-defined `and` and `or` and to operators defined by renaming declarations are not flagged. Calls to predefined `and` and `or` operators for modular types or boolean array types are not flagged.

This rule has no parameters.

### 1.5.9 OTHERS\_In\_Aggregates

Flag each use of an `others` choice in extension aggregates. In record and array aggregates, an `others` choice is flagged unless it is used to refer to all components, or to all but one component.

If, in case of a named array aggregate, there are two associations, one with an `others` choice and another with a discrete range, the `others` choice is flagged even if the discrete range specifies exactly one component; for example, `(1..1 => 0, others => 1)`.

This rule has no parameters.

### 1.5.10 OTHERS\_In\_CASE\_Statements

Flag any use of an `others` choice in a case statement.

This rule has no parameters.

### 1.5.11 OTHERS\_In\_Exception\_Handlers

Flag any use of an `others` choice in an exception handler.

This rule has no parameters.

### 1.5.12 Overly\_Nested\_Control\_Structures

Flag each control structure whose nesting level exceeds the value provided in the rule parameter.

The control structures checked are the following:

- `if` statement

- case statement
- loop statement
- Selective accept statement
- Timed entry call statement
- Conditional entry call
- Asynchronous select statement

The rule has the following parameter for the ‘+R’ option:

*N*            Positive integer specifying the maximal control structure nesting level that is not flagged

If the parameter for the ‘+R’ option is not specified or if it is not a positive integer, ‘+R’ option is ignored.

If more then one option is specified for the gnatcheck call, the later option and new parameter override the previous one(s).

#### 1.5.13 Positional\_Actuals\_For\_Defaulted\_Generic\_Parameters

Flag each generic actual parameter corresponding to a generic formal parameter with a default initialization, if positional notation is used.

This rule has no parameters.

#### 1.5.14 Positional\_Actuals\_For\_Defaulted\_Parameters

Flag each actual parameter to a subprogram or entry call where the corresponding formal parameter has a default expression, if positional notation is used.

This rule has no parameters.

#### 1.5.15 Positional\_Components

Flag each array, record and extension aggregate that includes positional notation.

This rule has no parameters.

#### 1.5.16 Positional\_Generic\_Parameters

Flag each positional actual generic parameter except for the case when the generic unit being instantiated has exactly one generic formal parameter.

This rule has no parameters.

### 1.5.17 Positional\_Parameters

Flag each positional parameter notation in a subprogram or entry call, except for the following:

- Parameters of calls to of prefix or infix operators are not flagged
- If the called subprogram or entry has only one formal parameter, the parameter of the call is not flagged;
- If a subprogram call uses the *Object.Operation* notation, then
  - the first parameter (that is, *Object*) is not flagged;
  - if the called subprogram has only two parameters, the second parameter of the call is not flagged;

This rule has no parameters.

### 1.5.18 Unconditional\_Exits

Flag unconditional `exit` statements.

This rule has no parameters.

### 1.5.19 Unnamed\_Blocks\_And\_Loops

Flag each unnamed block statement and loop statement.

The rule has no parameters.

### 1.5.20 USE\_PACKAGE\_Clauses

Flag all use clauses for packages; `use type` clauses are not flagged.

This rule has no parameters.

## 1.6 Readability

The rules described in this section may be used to enforce feature usages that contribute towards readability.

### 1.6.1 Misnamed\_Controlling\_Parameters

Flags a declaration of a dispatching operation, if the first parameter is not a controlling one and its name is not `This` (the check for parameter name is not case-sensitive). Declarations of dispatching functions with controlling result and no controlling parameter are never flagged.

A subprogram body declaration, subprogram renaming declaration or subprogram body stub is flagged only if it is not a completion of a prior subprogram declaration.

This rule has no parameters.

### 1.6.2 Misnamed\_Identifiers

Flag the declaration of each identifier that does not have a suffix corresponding to the kind of entity being declared. The following declarations are checked:

- type declarations
- subtype declarations
- constant declarations (but not number declarations)
- package renaming declarations (but not generic package renaming declarations)

This rule may have parameters. When used without parameters, the rule enforces the following checks:

- type-defining names end with `_T`, unless the type is an access type, in which case the suffix must be `_A`
- constant names end with `_C`
- names defining package renamings end with `_R`

Defining identifiers from incomplete type declarations are never flagged.

For a private type declaration (including private extensions), the defining identifier from the private type declaration is checked against the type suffix (even if the corresponding full declaration is an access type declaration), and the defining identifier from the corresponding full type declaration is not checked.

For a deferred constant, the defining name in the corresponding full constant declaration is not checked.

Defining names of formal types are not checked.

The rule may have the following parameters:

- For the ‘+R’ option:

Default     Sets the default listed above for all the names to be checked.

Type\_Suffix=*string*

Specifies the suffix for a type name.

Access\_Suffix=*string*

Specifies the suffix for an access type name. If this parameter is set, it overrides for access types the suffix set by the Type\_Suffix parameter. For access types, *string* may have the following format: *suffix1(suffix2)*. That means that an access type name should have the *suffix1* suffix except for the case when the designated type is also an access type, in this case the type name should have the *suffix1 & suffix2* suffix.

Class\_Access\_Suffix=*string*

Specifies the suffix for the name of an access type that points to some class-wide type. If this parameter is set, it overrides for

such access types the suffix set by the `Type_Suffix` or `Access_Suffix` parameter.

`Class_Subtype_Suffix=string`

Specifies the suffix for the name of a subtype that denotes a class-wide type.

`Constant_Suffix=string`

Specifies the suffix for a constant name.

`Renaming_Suffix=string`

Specifies the suffix for a package renaming name.

- For the ‘-R’ option:

`All_Suffixes`

Remove all the suffixes specified for the identifier suffix checks, whether by default or as specified by other rule parameters. All the checks for this rule are disabled as a result.

`Type_Suffix`

Removes the suffix specified for types. This disables checks for types but does not disable any other checks for this rule (including the check for access type names if `Access_Suffix` is set).

`Access_Suffix`

Removes the suffix specified for access types. This disables checks for access type names but does not disable any other checks for this rule. If `Type_Suffix` is set, access type names are checked as ordinary type names.

`Class_Access_Suffix`

Removes the suffix specified for access types pointing to class-wide type. This disables specific checks for names of access types pointing to class-wide types but does not disable any other checks for this rule. If `Type_Suffix` is set, access type names are checked as ordinary type names. If `Access_Suffix` is set, these access types are checked as any other access type name.

`Class_Subtype_Suffix=string`

Removes the suffix specified for subtype names. This disables checks for subtype names but does not disable any other checks for this rule.

`Constant_Suffix`

Removes the suffix specified for constants. This disables checks for constant names but does not disable any other checks for this rule.



`Renaming_Suffix`

Removes the suffix specified for package renamings. This disables checks for package renamings but does not disable any other checks for this rule.

If more than one parameter is used, parameters must be separated by commas.

If more than one option is specified for the `gnatcheck` invocation, a new option overrides the previous one(s).

The `'+RMisnamed_Identifiers'` option (with no parameter) enables checks for all the name suffixes specified by previous options used for this rule.

The `'-RMisnamed_Identifiers'` option (with no parameter) disables all the checks but keeps all the suffixes specified by previous options used for this rule.

The *string* value must be a valid suffix for an Ada identifier (after trimming all the leading and trailing space characters, if any). Parameters are not case sensitive, except the *string* part.

If any error is detected in a rule parameter, the parameter is ignored. In such a case the options that are set for the rule are not specified.

### 1.6.3 `Name_Clashes`

Check that certain names are not used as defining identifiers. To activate this rule, you need to supply a reference to the dictionary file(s) as a rule parameter(s) (more than one dictionary file can be specified). If no dictionary file is set, this rule will not cause anything to be flagged. Only defining occurrences, not references, are checked. The check is not case-sensitive.

This rule is enabled by default, but without setting any corresponding dictionary file(s); thus the default effect is to do no checks.

A dictionary file is a plain text file. The maximum line length for this file is 1024 characters. If the line is longer than this limit, extra characters are ignored.

Each line can be either an empty line, a comment line, or a line containing a list of identifiers separated by space or HT characters. A comment is an Ada-style comment (from `--` to end-of-line). Identifiers must follow the Ada syntax for identifiers. A line containing one or more identifiers may end with a comment.

### 1.6.4 `Uncommented_BEGIN_In_Package_Bodies`

Flags each package body with declarations and a statement part that does not include a trailing comment on the line containing the `begin` keyword; this trailing comment needs to specify the package name and nothing else. The `begin` is not flagged if the package body does not contain any declarations.

If the `begin` keyword is placed on the same line as the last declaration or the first statement, it is flagged independently of whether the line contains a trailing comment. The diagnostic message is attached to the line containing the first statement.

This rule has no parameters.

## 1.7 Source Code Presentation

This section is a placeholder; there are currently no rules in this category.

## 2 Feature Usage Rules

The rules in this chapter can be used to enforce specific usage patterns for a variety of language features.

### 2.1 Abstract\_Type\_Declarations

Flag all declarations of abstract types. For an abstract private type, both the private and full type declarations are flagged.

This rule has no parameters.

### 2.2 Anonymous\_Subtypes

Flag all uses of anonymous subtypes (except cases when subtype indication is a part of a record component definition, and this subtype indication depends on a discriminant). A use of an anonymous subtype is any instance of a subtype indication with a constraint, other than one that occurs immediately within a subtype declaration. Any use of a range other than as a constraint used immediately within a subtype declaration is considered as an anonymous subtype.

An effect of this rule is that `for` loops such as the following are flagged (since `1..N` is formally a “range”):

```
for I in 1 .. N loop
  ...
end loop;
```

Declaring an explicit subtype solves the problem:

```
subtype S is Integer range 1..N;
...
for I in S loop
  ...
end loop;
```

This rule has no parameters.

### 2.3 Blocks

Flag each block statement.

This rule has no parameters.

### 2.4 Complex\_Inlined\_Subprograms

Flags a subprogram (or generic subprogram) if pragma `Inline` is applied to the subprogram and at least one of the following conditions is met:

- it contains at least one complex declaration such as a subprogram body, package, task, protected declaration, or a generic instantiation (except instantiation of `Ada.Unchecked_Conversion`);

- it contains at least one complex statement such as a loop, a case or a if statement, or a short circuit control form;
- the number of statements exceeds a value specified by the ‘N’ rule parameter;

This rule has the following (mandatory) parameter for the ‘+R’ option:

*N*            Positive integer specifying the maximum allowed total number of statements in the subprogram body.

## 2.5 Controlled\_Type\_Declarations

Flag all declarations of controlled types. A declaration of a private type is flagged if its full declaration declares a controlled type. A declaration of a derived type is flagged if its ancestor type is controlled. Subtype declarations are not checked. A declaration of a type that itself is not a descendant of a type declared in `Ada.Finalization` but has a controlled component is not checked.

This rule has no parameters.

## 2.6 Declarations\_In\_Blocks

Flag all block statements containing local declarations. A `declare` block with an empty *declarative\_part* or with a *declarative part* containing only pragmas and/or use clauses is not flagged.

This rule has no parameters.

## 2.7 Deeply\_Nested\_Inlining

Flags a subprogram (or generic subprogram) if pragma `Inline` has been applied to the subprogram but the subprogram calls to another inlined subprogram that results in nested inlining with nesting depth exceeding the value specified by the ‘N’ rule parameter.

This rule requires the global analysis of all the compilation units that are `gnatcheck` arguments; such analysis may affect the tool’s performance.

This rule has the following (mandatory) parameter for the ‘+R’ option:

*N*            Positive integer specifying the maximal allowed level of nested inlining.

## 2.8 Default\_Parameters

Flag all default expressions for subprogram parameters. Parameter declarations of formal and generic subprograms are also checked.

This rule has no parameters.

## 2.9 Discriminated\_Records

Flag all declarations of record types with discriminants. Only the declarations of record and record extension types are checked. Incomplete, formal, private, derived and private extension type declarations are not checked. Task and protected type declarations also are not checked.

This rule has no parameters.

## 2.10 Explicit\_Full\_Discrete\_Ranges

Flag each discrete range that has the form `A'First .. A'Last`.

This rule has no parameters.

## 2.11 Float\_Equality\_Checks

Flag all calls to the predefined equality operations for floating-point types. Both “=” and “/=” operations are checked. User-defined equality operations are not flagged, nor are “=” and “/=” operations for fixed-point types.

This rule has no parameters.

## 2.12 Function\_Style\_Procedures

Flag each procedure that can be rewritten as a function. A procedure can be converted into a function if it has exactly one parameter of mode `out` and no parameters of mode `in out`. Procedure declarations, formal procedure declarations, and generic procedure declarations are always checked. Procedure bodies and body stubs are flagged only if they do not have corresponding separate declarations. Procedure renamings and procedure instantiations are not flagged.

If a procedure can be rewritten as a function, but its `out` parameter is of a limited type, it is not flagged.

Protected procedures are not flagged. Null procedures also are not flagged.

This rule has no parameters.

## 2.13 Generics\_In\_Subprograms

Flag each declaration of a generic unit in a subprogram. Generic declarations in the bodies of generic subprograms are also flagged. A generic unit nested in another generic unit is not flagged. If a generic unit is declared in a local package that is declared in a subprogram body, the generic unit is flagged.

This rule has no parameters.

## 2.14 Implicit\_IN\_Mode\_Parameters

Flag each occurrence of a formal parameter with an implicit `in` mode. Note that access parameters, although they technically behave like `in` parameters, are not flagged.

This rule has no parameters.

## 2.15 Improperly\_Located\_Instantiations

Flag all generic instantiations in library-level package specs (including library generic packages) and in all subprogram bodies.

Instantiations in task and entry bodies are not flagged. Instantiations in the bodies of protected subprograms are flagged.

This rule has no parameters.

## 2.16 Library\_Level\_Subprograms

Flag all library-level subprograms (including generic subprogram instantiations).

This rule has no parameters.

## 2.17 Non\_Qualified\_Aggregates

Flag each non-qualified aggregate. A non-qualified aggregate is an aggregate that is not the expression of a qualified expression. A string literal is not considered an aggregate, but an array aggregate of a string type is considered as a normal aggregate. Aggregates of anonymous array types are not flagged.

This rule has no parameters.

## 2.18 Numeric\_Literals

Flag each use of a numeric literal in an index expression, and in any circumstance except for the following:

- a literal occurring in the initialization expression for a constant declaration or a named number declaration, or
- an integer literal that is less than or equal to a value specified by the '`N`' rule parameter.

This rule may have the following parameters for the '`+R`' option:

<code>N</code>	<code>N</code> is an integer literal used as the maximal value that is not flagged (i.e., integer literals not exceeding this value are allowed)
<code>ALL</code>	All integer literals are flagged

If no parameters are set, the maximum unflagged value is 1.

The last specified check limit (or the fact that there is no limit at all) is used when multiple ‘+R’ options appear.

The ‘-R’ option for this rule has no parameters. It disables the rule but retains the last specified maximum unflagged value. If the ‘+R’ option subsequently appears, this value is used as the threshold for the check.

## 2.19 Parameters\_Out\_Of\_Order

Flag each subprogram and entry declaration whose formal parameters are not ordered according to the following scheme:

- in and access parameters first, then in out parameters, and then out parameters;
- for in mode, parameters with default initialization expressions occur last

Only the first violation of the described order is flagged.

The following constructs are checked:

- subprogram declarations (including null procedures);
- generic subprogram declarations;
- formal subprogram declarations;
- entry declarations;
- subprogram bodies and subprogram body stubs that do not have separate specifications

Subprogram renamings are not checked.

This rule has no parameters.

## 2.20 Raising\_Predefined\_Exceptions

Flag each raise statement that raises a predefined exception (i.e., one of the exceptions `Constraint_Error`, `Numeric_Error`, `Program_Error`, `Storage_Error`, or `Tasking_Error`).

This rule has no parameters.

## 2.21 Unassigned\_OUT\_Parameters

Flags procedures’ out parameters that are not assigned, and identifies the contexts in which the assignments are missing.

An out parameter is flagged in the statements in the procedure body’s handled sequence of statements (before the procedure body’s exception part, if any) if this sequence of statements contains no assignments to the parameter.

An `out` parameter is flagged in an exception handler in the exception part of the procedure body's handled sequence of statements if the handler contains no assignment to the parameter.

Bodies of generic procedures are also considered.

The following are treated as assignments to an `out` parameter:

- an assignment statement, with the parameter or some component as the target;
- passing the parameter (or one of its components) as an `out` or `in out` parameter.

This rule does not have any parameters.

## **2.22 Unconstrained\_Array\_Returns**

Flag each function returning an unconstrained array. Function declarations, function bodies (and body stubs) having no separate specifications, and generic function instantiations are checked. Function calls and function renamings are not checked.

Generic function declarations, and function declarations in generic packages are not checked, instead this rule checks the results of generic instantiations (that is, expanded specification and expanded body corresponding to an instantiation).

This rule has no parameters.



## 3 Metrics-Related Rules

The rules in this chapter can be used to enforce compliance with specific code metrics, by checking that the metrics computed for a program lie within user-specifiable bounds. Depending on the metric, there may be a lower bound, an upper bound, or both. A construct is flagged if the value of the metric exceeds the upper bound or is less than the lower bound.

The name of any metrics rule consists of the prefix `Metrics_` followed by the name of the corresponding metric: `Essential_Complexity`, `Cyclomatic_Complexity`, or `LSLOC`. (The “LSLOC” acronym stands for “Logical Source Lines Of Code”.) The meaning and the computed values of the metrics are the same as in `gnatmetric`.

For the ‘+R’ option, each metrics rule has a numeric parameter specifying the bound (integer or real, depending on a metric). The ‘-R’ option for the metrics rules does not have a parameter.

*Example:* the rule

```
+RMetrics_Cyclomatic_Complexity : 7
```

means that all bodies with cyclomatic complexity exceeding 7 will be flagged.

To turn OFF the check for cyclomatic complexity metric, use the following option:

```
-RMetrics_Cyclomatic_Complexity
```

### 3.1 Metrics\_Essential\_Complexity

The `Metrics_Essential_Complexity` rule takes a positive integer as upper bound. A construct exceeding this limit will be flagged.

### 3.2 Metrics\_Cyclomatic\_Complexity

The `Metrics_Cyclomatic_Complexity` rule takes a positive integer as upper bound. A construct exceeding this limit will be flagged.

### 3.3 Metrics\_LSLOC

The `Metrics_LSLOC` rule takes a positive integer as upper bound. A compilation unit exceeding this limit will be flagged.



## 4 SPARK Ada Rules

The rules in this chapter can be used to enforce compliance with the Ada subset allowed by the SPARK tools.

### 4.1 Boolean\_Relational\_Operators

Flag each call to a predefined relational operator (“<”, “>”, “<=”, “>=”, “=” and “/=”) for the predefined Boolean type. (This rule is useful in enforcing the SPARK language restrictions.)

Calls to predefined relational operators of any type derived from `Standard.Boolean` are not detected. Calls to user-defined functions with these designators, and uses of operators that are renamings of the predefined relational operators for `Standard.Boolean`, are likewise not detected.

This rule has no parameters.

### 4.2 Expanded\_Loop\_Exit\_Names

Flag all expanded loop names in `exit` statements.

This rule has no parameters.

### 4.3 Non\_SPARK\_Attributes

The SPARK language defines the following subset of Ada 95 attribute designators as those that can be used in SPARK programs. The use of any other attribute is flagged.

- ‘Adjacent
- ‘Aft
- ‘Base
- ‘Ceiling
- ‘Component\_Size
- ‘Compose
- ‘Copy\_Sign
- ‘Delta
- ‘Denorm
- ‘Digits
- ‘Exponent
- ‘First
- ‘Floor
- ‘Fore

- 'Fraction
- 'Last
- 'Leading\_Part
- 'Length
- 'Machine
- 'Machine\_Emax
- 'Machine\_Emin
- 'Machine\_Mantissa
- 'Machine\_Overflows
- 'Machine\_Radix
- 'Machine\_Rounds
- 'Max
- 'Min
- 'Model
- 'Model\_Emin
- 'Model\_Epsilon
- 'Model\_Mantissa
- 'Model\_Small
- 'Modulus
- 'Pos
- 'Pred
- 'Range
- 'Remainder
- 'Rounding
- 'Safe\_First
- 'Safe\_Last
- 'Scaling
- 'Signed\_Zeros
- 'Size
- 'Small
- 'Succ
- 'Truncation
- 'Unbiased\_Rounding
- 'Val
- 'Valid

This rule has no parameters.

#### 4.4 Non\_Tagged\_Derived\_Types

Flag all derived type declarations that do not have a record extension part.

This rule has no parameters.

#### 4.5 Outer\_Loop\_Exits

Flag each `exit` statement containing a loop name that is not the name of the immediately enclosing `loop` statement.

This rule has no parameters.

#### 4.6 Overloaded\_Operators

Flag each function declaration that overloads an operator symbol. A function body is checked only if the body does not have a separate spec. Formal functions are also checked. For a renaming declaration, only renaming-as-declaration is checked

This rule has no parameters.

#### 4.7 Slices

Flag all uses of array slicing

This rule has no parameters.

#### 4.8 Universal\_Ranges

Flag discrete ranges that are a part of an index constraint, constrained array definition, or `for`-loop parameter specification, and whose bounds are both of type *universal\_integer*. Ranges that have at least one bound of a specific type (such as `1 .. N`, where `N` is a variable or an expression of non-universal type) are not flagged.

This rule has no parameters.



## Appendix A List of Rules

This Appendix contains an alphabetized list of all the predefined GNATcheck rules.

- `Abstract_Type_Declarations`  
See [Section 2.1 \[Abstract\\_Type\\_Declarations\]](#), page 17.
- `Anonymous_Arrays`  
See [Section 1.5.1 \[Anonymous\\_Arrays\]](#), page 9.
- `Anonymous_Subtypes`  
See [Section 2.2 \[Anonymous\\_Subtypes\]](#), page 17.
- `Blocks`  
See [Section 2.3 \[Blocks\]](#), page 17.
- `Boolean_Relational_Operators`  
See [Section 4.1 \[Boolean\\_Relational\\_Operators\]](#), page 25.
- `Complex_Inlined_Subprograms`  
See [Section 2.4 \[Complex\\_Inlined\\_Subprograms\]](#), page 17.
- `Controlled_Type_Declarations`  
See [Section 2.5 \[Controlled\\_Type\\_Declarations\]](#), page 18.
- `Declarations_In_Blocks`  
See [Section 2.6 \[Declarations\\_In\\_Blocks\]](#), page 18.
- `Deep_Inheritance_Hierarchies`  
See [Section 1.2.1 \[Deep\\_Inheritance\\_Hierarchies\]](#), page 3.
- `Deeply_Nested_Generics`  
See [Section 1.4.1 \[Deeply\\_Nested\\_Generics\]](#), page 7.
- `Deeply_Nested_Inlining`  
See [Section 2.7 \[Deeply\\_Nested\\_Inlining\]](#), page 18.
- `Default_Parameters`  
See [Section 2.8 \[Default\\_Parameters\]](#), page 18.
- `Direct_Calls_To_Primitives`  
See [Section 1.2.2 \[Direct\\_Calls\\_To\\_Primitives\]](#), page 4.
- `Discriminated_Records`  
See [Section 2.9 \[Discriminated\\_Records\]](#), page 19.
- `Enumeration_Ranges_In_CASE_Statements`  
See [Section 1.5.2 \[Enumeration\\_Ranges\\_In\\_CASE\\_Statements\]](#), page 9.
- `Exceptions_As_Control_Flow`  
See [Section 1.5.3 \[Exceptions\\_As\\_Control\\_Flow\]](#), page 9.
- `Exits_From_Conditional_Loops`  
See [Section 1.5.4 \[Exits\\_From\\_Conditional\\_Loops\]](#), page 9.

- `EXIT_Statements_With_No_Loop_Name`  
See [Section 1.5.5 \[EXIT\\_Statements\\_With\\_No\\_Loop\\_Name\]](#), page 9.
- `Expanded_Loop_Exit_Names`  
See [Section 4.2 \[Expanded\\_Loop\\_Exit\\_Names\]](#), page 25.
- `Explicit_Full_Discrete_Ranges`  
See [Section 2.10 \[Explicit\\_Full\\_Discrete\\_Ranges\]](#), page 19.
- `Float_Equality_Checks`  
See [Section 2.11 \[Float\\_Equality\\_Checks\]](#), page 19.
- `Forbidden_Attributes`  
See [Section 1.3.1 \[Forbidden\\_Attributes\]](#), page 4.
- `Forbidden_Pragmas`  
See [Section 1.3.2 \[Forbidden\\_Pragmas\]](#), page 6.
- `Function_Style_Procedures`  
See [Section 2.12 \[Function\\_Style\\_Procedures\]](#), page 19.
- `Generics_In_Subprograms`  
See [Section 2.13 \[Generics\\_In\\_Subprograms\]](#), page 19.
- `GOTO_Statements`  
See [Section 1.5.6 \[GOTO\\_Statements\]](#), page 9.
- `Implicit_IN_Mode_Parameters`  
See [Section 2.14 \[Implicit\\_IN\\_Mode\\_Parameters\]](#), page 20.
- `Implicit_SMALL_For_Fixed_Point_Types`  
See [Section 1.3.3 \[Implicit\\_SMALL\\_For\\_Fixed\\_Point\\_Types\]](#), page 7.
- `Improperly_Located_Instantiations`  
See [Section 2.15 \[Improperly\\_Located\\_Instantiations\]](#), page 20.
- `Improper_Returns`  
See [Section 1.5.7 \[Improper\\_Returns\]](#), page 9.
- `Library_Level_Subprograms`  
See [Section 2.16 \[Library\\_Level\\_Subprograms\]](#), page 20.
- `Local_Packages`  
See [Section 1.4.2 \[Local\\_Packages\]](#), page 8.
- `Metrics_Cyclomatic_Complexity`  
See [Section 3.2 \[Metrics\\_Cyclomatic\\_Complexity\]](#), page 23.
- `Metrics_Essential_Complexity`  
See [Section 3.1 \[Metrics\\_Essential\\_Complexity\]](#), page 23.
- `Metrics_LSLLOC`  
See [Section 3.3 \[Metrics\\_LSLLOC\]](#), page 23.
- `Misnamed_Controlling_Parameters`  
See [Section 1.6.1 \[Misnamed\\_Controlling\\_Parameters\]](#), page 12.



- `Misnamed_Identifiers`  
See [Section 1.6.2 \[Misnamed\\_Identifiers\]](#), page 13.
- `Multiple_Entries_In_Protected_Definitions`  
See [Section 1.1.1 \[Multiple\\_Entries\\_In\\_Protected\\_Definitions\]](#), page 3.
- `Name_Clashes`  
See [Section 1.6.3 \[Name\\_Clashes\]](#), page 15.
- `Non_Qualified_Aggregates`  
See [Section 2.17 \[Non\\_Qualified\\_Aggregates\]](#), page 20.
- `Non_Short_Circuit_Operators`  
See [Section 1.5.8 \[Non\\_Short\\_Circuit\\_Operators\]](#), page 10.
- `Non_SPARK_Attributes`  
See [Section 4.3 \[Non\\_SPARK\\_Attributes\]](#), page 25.
- `Non_Tagged_Derived_Types`  
See [Section 4.4 \[Non\\_Tagged\\_Derived\\_Types\]](#), page 27.
- `Non_Visible_Exceptions`  
See [Section 1.4.3 \[Non\\_Visible\\_Exceptions\]](#), page 8.
- `Numeric_Literals`  
See [Section 2.18 \[Numeric\\_Literals\]](#), page 20.
- `OTHERS_In_Aggregates`  
See [Section 1.5.9 \[OTHERS\\_In\\_Aggregates\]](#), page 10.
- `OTHERS_In_CASE_Statements`  
See [Section 1.5.10 \[OTHERS\\_In\\_CASE\\_Statements\]](#), page 10.
- `OTHERS_In_Exception_Handlers`  
See [Section 1.5.11 \[OTHERS\\_In\\_Exception\\_Handlers\]](#), page 10.
- `Outer_Loop_Exits`  
See [Section 4.5 \[Outer\\_Loop\\_Exits\]](#), page 27.
- `Overloaded_Operators`  
See [Section 4.6 \[Overloaded\\_Operators\]](#), page 27.
- `Overly_Nested_Control_Structures`  
See [Section 1.5.12 \[Overly\\_Nested\\_Control\\_Structures\]](#), page 10.
- `Parameters_Out_Of_Order`  
See [Section 2.19 \[Parameters\\_Out\\_Of\\_Order\]](#), page 21.
- `Positional_Actuals_For_Defaulted_Generic_Parameters`  
See [Section 1.5.13 \[Positional\\_Actuals\\_For\\_Defaulted\\_Generic\\_Parameters\]](#), page 11.
- `Positional_Actuals_For_Defaulted_Parameters`  
See [Section 1.5.14 \[Positional\\_Actuals\\_For\\_Defaulted\\_Parameters\]](#), page 11.

- `Positional_Components`  
See [Section 1.5.15 \[Positional\\_Components\]](#), page 11.
- `Positional_Generic_Parameters`  
See [Section 1.5.16 \[Positional\\_Generic\\_Parameters\]](#), page 11.
- `Positional_Parameters`  
See [Section 1.5.17 \[Positional\\_Parameters\]](#), page 12.
- `Predefined_Numeric_Types`  
See [Section 1.3.4 \[Predefined\\_Numeric\\_Types\]](#), page 7.
- `Raising_External_Exceptions`  
See [Section 1.4.4 \[Raising\\_External\\_Exceptions\]](#), page 8.
- `Raising_Predefined_Exceptions`  
See [Section 2.20 \[Raising\\_Predefined\\_Exceptions\]](#), page 21.
- `Separate_Numeric_Error_Handlers`  
See [Section 1.3.5 \[Separate\\_Numeric\\_Error\\_Handlers\]](#), page 7.
- `Slices`  
See [Section 4.7 \[Slices\]](#), page 27.
- `Too_Many_Parents`  
See [Section 1.2.3 \[Too\\_Many\\_Parents\]](#), page 4.
- `Unassigned_OUT_Parameters`  
See [Section 2.21 \[Unassigned\\_OUT\\_Parameters\]](#), page 21.
- `Uncommented_BEGIN_In_Package_Bodies`  
See [Section 1.6.4 \[Uncommented\\_BEGIN\\_In\\_Package\\_Bodies\]](#), page 15.
- `Unconditional_Exits`  
See [Section 1.5.18 \[Unconditional\\_Exits\]](#), page 12.
- `Unconstrained_Array_Returns`  
See [Section 2.22 \[Unconstrained\\_Array\\_Returns\]](#), page 22.
- `Universal_Ranges`  
See [Section 4.8 \[Universal\\_Ranges\]](#), page 27.
- `Unnamed_Blocks_And_Loops`  
See [Section 1.5.19 \[Unnamed\\_Blocks\\_And\\_Loops\]](#), page 12.
- `USE_PACKAGE_Clauses`  
See [Section 1.5.20 \[USE\\_PACKAGE\\_Clauses\]](#), page 12.
- `Visible_Components`  
See [Section 1.2.4 \[Visible\\_Components\]](#), page 4.
- `Volatile_Objects_Without_Address_Clauses`  
See [Section 1.1.2 \[Volatile\\_Objects\\_Without\\_Address\\_Clauses\]](#), page 3.

# Appendix B GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.  
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related

matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

## **2. VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading

or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### **3. COPYING IN QUANTITY**

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified

Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise



combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

#### Heading 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

### **7. AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

### **8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

### **9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify,



sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.



# Index

## A

Abstract_Type_Declarations .....	17
Anonymous_Arrays .....	9
Anonymous_Subtypes .....	17

## B

Blocks .....	17
Boolean_Relational_Operators .....	25

## C

Complex_Inlined_Subprograms .....	17
Controlled_Type_Declarations .....	18

## D

Declarations_In_Blocks .....	18
Deep_Inheritance_Hierarchies .....	3
Deeply_Nested_Generics .....	7
Deeply_Nested_Inlining .....	18
Default_Parameters .....	18
Direct_Calls_To_Primitives .....	4
Discriminated_Records .....	19

## E

Enumeration_Ranges_In_CASE_Statements .....	9
Exceptions_As_Control_Flow .....	9
EXIT_Statements_With_No_Loop_Name .....	9
Exits_From_Conditional_Loops .....	9
Expanded_Loop_Exit_Names .....	25
Explicit_Full_Discrete_Ranges .....	19

## F

Feature usage related rules .....	17
Float_Equality_Checks .....	19
Forbidden_Attributes .....	4
Forbidden_Pragmas .....	6
Free Documentation License, GNU .....	33
Function_Style_Procedures .....	19

## G

Generics_In_Subprograms .....	19
GNU Free Documentation License .....	33
GOTO_Statements .....	9

## I

Implicit_IN_Mode_Parameters .....	20
Implicit_SMALL_For_Fixed_Point_Types ..	7
Improper_Returns .....	10
Improperly_Located_Instantiations .....	20

## L

Library_Level_Subprograms .....	20
License, GNU Free Documentation .....	33
Local_Packages .....	8

## M

Metrics-related rules .....	23
Metrics_Cyclomatic_Complexity .....	23
Metrics_Essential_Complexity .....	23
Metrics_LSLLOC .....	23
Misnamed_Controlling_Parameters .....	12
Misnamed_Identifiers .....	13
Multiple_Entries_In_Protected_Definitions .....	3

## N

Name_Clashes .....	15
Non_Qualified_Aggregates .....	20
Non_Short_Circuit_Operators .....	10
Non_SPARK_Attributes .....	25
Non_Tagged_Derived_Types .....	27
Non_Visible_Exceptions rule .....	8
Numeric_Literals .....	20

**O**

Object-Orientation related rules.....	3
OTHERS_In_Aggregates.....	10
OTHERS_In_CASE_Statements.....	10
OTHERS_In_Exception_Handlers.....	10
Outer_Loop_Exits.....	27
Overloaded_Operators.....	27
Overly_Nested_Control_Structures.....	10

**P**

Parameters_Out_Of_Order.....	21
Portability-related rules.....	4
Positional_Actuals_For_Defaulted_ Generic_Parameters.....	11
Positional_Actuals_For_Defaulted_ Parameters rule.....	11
Positional_Components.....	11
Positional_Generic_Parameters.....	11
Positional_Parameters.....	12
Predefined_Numeric_Types.....	7
Program Structure related rules.....	7
Programming Practice related rules.....	8

**R**

Raising_External_Exceptions.....	8
Raising_Predefined_Exceptions.....	21
Readability-related rules.....	12

**S**

Separate_Numeric_Error_Handlers.....	7
Slices.....	27
Source code presentation related rules.....	16
SPARK Ada related rules.....	25
Style-related rules.....	3

**T**

Tasking-related rules.....	3
Too_Many_Parents.....	4

**U**

Unassigned_OUT_Parameters.....	21
Uncommented_BEGIN_In_Package_Bodies ..	15
Unconditional_Exits rule.....	12
Unconstrained_Array_Returns.....	22
Universal_Ranges rule.....	27
Unnamed_Blocks_And_Loops.....	12
USE_PACKAGE_Clauses.....	12

**V**

Visible_Components.....	4
Volatile_Objects_Without_Address_Clauses .....	3

---

# Table of Contents

<b>About This Manual</b>	<b>1</b>
What This Guide Contains	1
What You Should Know Before Reading This Guide	1
<b>1 Style-Related Rules</b>	<b>3</b>
1.1 Tasking	3
1.1.1 Multiple_Entries_In_Protected_Definitions	3
1.1.2 Volatile_Objects_Without_Address_Clauses	3
1.2 Object Orientation	3
1.2.1 Deep_Inheritance_Hierarchies	3
1.2.2 Direct_Calls_To_Primitives	4
1.2.3 Too_Many_Parents	4
1.2.4 Visible_Components	4
1.3 Portability	4
1.3.1 Forbidden_Attributes	4
1.3.2 Forbidden_Pragmas	6
1.3.3 Implicit_SMALL_For_Fixed_Point_Types	7
1.3.4 Predefined_Numeric_Types	7
1.3.5 Separate_Numeric_Error_Handlers	7
1.4 Program Structure	7
1.4.1 Deeply_Nested_Generics	7
1.4.2 Local_Packages	8
1.4.3 Non_Visible_Exceptions	8
1.4.4 Raising_External_Exceptions	8
1.5 Programming Practice	8
1.5.1 Anonymous_Arrays	9
1.5.2 Enumeration_Ranges_In_CASE_Statements	9
1.5.3 Exceptions_As_Control_Flow	9
1.5.4 Exits_From_Conditional_Loops	9
1.5.5 EXIT_Statements_With_No_Loop_Name	9
1.5.6 GOTO_Statements	9
1.5.7 Improper_Returns	9
1.5.8 Non_Short_Circuit_Operators	10
1.5.9 OTHERS_In_Aggregates	10
1.5.10 OTHERS_In_CASE_Statements	10
1.5.11 OTHERS_In_Exception_Handlers	10
1.5.12 Overly_Nested_Control_Structures	10
1.5.13 Positional_Actuals_For_Defaulted_Generic_Parameters	11
.....	11

1.5.14	Positional_Actuals_For_Defaulted_Parameters .....	11
1.5.15	Positional_Components .....	11
1.5.16	Positional_Generic_Parameters .....	11
1.5.17	Positional_Parameters .....	12
1.5.18	Unconditional_Exits .....	12
1.5.19	Unnamed_Blocks_And_Loops .....	12
1.5.20	USE_PACKAGE_Clauses .....	12
1.6	Readability .....	12
1.6.1	Misnamed_Controlling_Parameters .....	12
1.6.2	Misnamed_Identifiers .....	13
1.6.3	Name_Clashes .....	15
1.6.4	Uncommented_BEGIN_In_Package_Bodies .....	15
1.7	Source Code Presentation .....	16
<b>2</b>	<b>Feature Usage Rules .....</b>	<b>17</b>
2.1	Abstract_Type_Declarations .....	17
2.2	Anonymous_Subtypes .....	17
2.3	Blocks .....	17
2.4	Complex_Inlined_Subprograms .....	17
2.5	Controlled_Type_Declarations .....	18
2.6	Declarations_In_Blocks .....	18
2.7	Deeply_Nested_Inlining .....	18
2.8	Default_Parameters .....	18
2.9	Discriminated_Records .....	19
2.10	Explicit_Full_Discrete_Ranges .....	19
2.11	Float_Equality_Checks .....	19
2.12	Function_Style_Procedures .....	19
2.13	Generics_In_Subprograms .....	19
2.14	Implicit_IN_Mode_Parameters .....	20
2.15	Improperly_Located_Instantiations .....	20
2.16	Library_Level_Subprograms .....	20
2.17	Non_Qualified_Aggregates .....	20
2.18	Numeric_Literals .....	20
2.19	Parameters_Out_Of_Order .....	21
2.20	Raising_Predefined_Exceptions .....	21
2.21	Unassigned_OUT_Parameters .....	21
2.22	Unconstrained_Array_Returns .....	22
<b>3</b>	<b>Metrics-Related Rules .....</b>	<b>23</b>
3.1	Metrics_Essential_Complexity .....	23
3.2	Metrics_Cyclomatic_Complexity .....	23
3.3	Metrics_LSLOC .....	23

---

<b>4</b>	<b>SPARK Ada Rules.....</b>	<b>25</b>
4.1	Boolean_Relational_Operators.....	25
4.2	Expanded_Loop_Exit_Names.....	25
4.3	Non_SPARK_Attributes.....	25
4.4	Non_Tagged_Derived_Types.....	27
4.5	Outer_Loop_Exits.....	27
4.6	Overloaded_Operators.....	27
4.7	Slices.....	27
4.8	Universal_Ranges.....	27
<b>Appendix A</b>	<b>List of Rules .....</b>	<b>29</b>
<b>Appendix B</b>	<b>GNU Free Documentation License....</b>	<b>33</b>
<b>Index.....</b>		<b>41</b>

