

# GNAT User's Guide for Native Platforms

---

GNAT User's Guide for Native Platforms , May 19, 2026

AdaCore

Copyright © 2008-2026, Free Software Foundation

---















‘GNAT, The GNU Ada Development Environment’

GCC version 17.0.0

AdaCore

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover Texts being “GNAT User’s Guide for Native Platforms”, and with no Back-Cover Texts. A copy of the license is included in the section entitled [GNU Free Documentation License], page 318.



## 1.2 What You Should Know before Reading This Guide

This guide assumes a basic familiarity with the Ada 95 language, as described in the International Standard ANSI/ISO/IEC-8652:1995, January 1995. Reference manuals for Ada 95, Ada 2005, Ada 2012 and Ada 2022 are included in the GNAT documentation package.

## 1.3 Related Information

For further information about Ada and related tools, please refer to the following documents:

- \* *Ada 95 Reference Manual*, *Ada 2005 Reference Manual*, *Ada 2012 Reference Manual*, and *Ada 2022 Reference Manual*, which contain reference material for the several revisions of the Ada language standard.
- \* *GNAT Reference Manual*, which contains all reference material for the GNAT implementation of Ada.
- \* *Using GNAT Studio*, which describes the GNAT Studio Integrated Development Environment.
- \* *GNAT Studio Tutorial*, which introduces the main GNAT Studio features through examples.
- \* *Debugging with GDB*, for all details on the use of the GNU source-level debugger.

## 1.4 Conventions

Following are examples of the typographical and graphic conventions used in this guide:

- \* Functions, utility program names, standard names, and classes.
- \* Option flags
- \* File names
- \* Variables
- \* ‘Emphasis’
- \* [optional information or parameters]
- \* Examples are described by text  
and then shown this way.
- \* Commands that you enter are shown as preceded by a prompt string comprising the \$ character followed by a space.
- \* Full file names are shown with the ‘/’ character as the directory separator; e.g., `parent-dir/subdir/myfile.adb`. If you are using GNAT on a Windows platform, please note that you should use the ‘\’ character instead.





```

end Greetings;

with Ada.Text_IO; use Ada.Text_IO;
package body Greetings is
  procedure Hello is
  begin
    Put_Line ("Hello WORLD!");
  end Hello;

  procedure Goodbye is
  begin
    Put_Line ("Goodbye WORLD!");
  end Goodbye;
end Greetings;

with Greetings;
procedure Gmain is
begin
  Greetings.Hello;
  Greetings.Goodbye;
end Gmain;

```

Following the one-unit-per-file rule, place this program in the following three separate files:

```

'greetings.ads'
    spec of package Greetings

'greetings.adb'
    body of package Greetings

'gmain.adb'
    body of main program

```

Note that there is no required order of compilation when using GNAT. In particular it is perfectly fine to compile the main program first. Also, it is not necessary to compile package specs in the case where there is an accompanying body; you only need compile the body. If you want to submit these files to the compiler for semantic checking and not code generation, use the `-gnatc` switch:

```
$ gcc -c greetings.ads -gnatc
```

Although you can do the compilation in separate steps, in practice it's almost always more convenient to use the `gnatmake` or `gprbuild` tools:

```
$ gnatmake gmain.adb
```





































use the `-w` switch, in which case the last occurrence in the last file will be the one that is output and `gnatchop` will skip earlier duplicate occurrences for the same unit.

### 3.4 Configuration Pragmas

Configuration pragmas supported by GNAT consist of those pragmas described as such in the Ada Reference Manual and the implementation-dependent pragmas that are configuration pragmas. See the `Implementation_Defined_Pragmas` chapter in the *GNAT-Reference-Manual* for details on these additional GNAT-specific configuration pragmas. Most notably, the pragma `Source_File_Name`, which allows specifying non-default names for source files, is a configuration pragma. The following is a complete list of configuration pragmas recognized by GNAT:

```
Ada_83
Ada_95
Ada_05
Ada_2005
Ada_12
Ada_2012
Ada_2022
Aggregate_Individually_Assign
Allow_Integer_Address
Annotate
Assertion_Policy
Assume_No_Invalid_Values
C_Pass_By_Copy
Check_Float_Overflow
Check_Name
Check_Policy
Component_Alignment
Convention_Identifier
Debug_Policy
Default_Scalar_Storage_Order
Default_Storage_Pool
Detect_Blocking
Disable_Atomic_Synchronization
Discard_Names
Elaboration_Checks
Eliminate
Enable_Atomic_Synchronization
Extend_System
Extensions_Allowed
External_Name_Casing
Fast_Math
Favor_Top_Level
Ignore_Pragma
Implicit_Packing
InitializeScalars
```

Interrupt\_State  
Interrupts\_System\_By\_Default  
License  
Locking\_Policy  
No\_Component\_Reordering  
No\_Heap\_Finalization  
No\_Strict\_Aliasing  
NormalizeScalars  
Optimize\_Alignment  
Overflow\_Mode  
Overriding\_Renamings  
Partition\_Elaboration\_Policy  
Persistent\_BSS  
Prefix\_Exception\_Messages  
Priority\_Specific\_Dispatching  
Profile  
Profile\_Warnings  
Queuing\_Policy  
Rename\_Pragma  
Restrictions  
Restriction\_Warnings  
Reviewable  
Short\_Circuit\_And\_Or  
Source\_File\_Name  
Source\_File\_Name\_Project  
SPARK\_Mode  
Style\_Checks  
Suppress  
Suppress\_Exception\_Locations  
Task\_Dispatching\_Policy  
Unevaluated\_Use\_Of\_Old  
Unsuppress  
Use\_VADS\_Size  
User\_Aspect\_Definition  
Validity\_Checks  
Warning\_As\_Error  
Warnings  
Wide\_Character\_Encoding

### 3.4.1 Handling of Configuration Pragas

You can place configuration pragmas either appear at the start of a compilation unit or in a configuration pragma file that applies to all compilations performed in a given compilation environment.

Configuration pragmas placed before a library level package specification are not propagated to the corresponding package body (see RM 10.1.5(8)); they must be added explicitly to the package body.







































































































































































































































































































































































































































































































































































































































