

# GNAT Studio

# Introduction

# GNAT Studio

- Our flagship IDE
- Available on Linux, Windows
- Supports native, cross, and bare-board development
  - Same look-and-feel
- Provides fully symbolic source-level debugging
- Supports Ada 2012 and all prior versions
- Supports C, C++ and Python

# GNAT Studio IDE

The screenshot displays the GNAT Studio IDE interface. The window title is "GNAT Studio - Welcome - - Default project". The menu bar includes File, Edit, Navigate, Find, Code, VCS, Build, SPARK, CodePeer, Analyze, Debug, View, Window, and Help. The toolbar contains icons for file operations, search, and navigation. The main workspace shows the GNAT Studio logo and version 21.7, along with a welcome message and links to user guides and tutorials. The left sidebar contains the Project, Scenario, and Outline views. The bottom panel shows the Messages and Locations views.

Callouts point to the following components:

- Tool Bar**: Points to the toolbar at the top left.
- Scenario / Project Views**: Points to the Project and Scenario views in the left sidebar.
- Outline / Learn Views**: Points to the Outline and Learn views in the left sidebar.
- Menu Bar**: Points to the menu bar at the top.
- Workspace**: Points to the main workspace area.
- Messages / Locations / Etc**: Points to the Messages and Locations views at the bottom.

# Integrated with GNAT Project Files

- Graphically presents what the project file specifies
  - Source directories
  - Relationships to other projects
  - Object and executable directories
  - Etc.
- Builds apps per the project file settings
  - Specified toolchain
  - Switches to be applied
- GUI for working with *scenario variables*

# GNAT Studio - Project Perspective

The screenshot shows the GNAT Studio interface in Project Perspective. On the left, a tree view displays the project structure. On the right, the main editor shows the source code for `screen_output.adb`. Below the editor are panels for Messages and Locations.

Callouts on the left side of the image explain the structure:

- Project node**: Points to the root of the project tree.
- Source\_Dirs entry**: Points to the `common` directory under the Project node.
- Files for specified languages**: Points to the `except.ads`, `input.adb`, and `input.ads` files under the `common` directory.
- Current file**: Points to the selected `screen_output.adb` file.
- Source\_Dirs entry**: Points to the `sdcs` directory under the `screen_output.adb` node.
- Obj\_Dir entry**: Points to the `obj` directory under the `screen_output.adb` node.
- Exec\_Dir entry**: Points to the `stack.adb` file under the `sdcs` directory.

The main editor displays the following Ada code:

```

1 with Ada.Text_IO; use Ada.Text_IO;
2 with Input;
3
4 package body Screen_Output is
5
6     -----
7     -- Local Data --
8     -----
9
10    Debug_On : constant Boolean := False;
11    -- When set, debugging information messages are output on the screen.
12
13    -----
14    -- Hg --
15    -----
16
17    procedure Hg
18      (S1 : String;
19       S2 : String := "");
20    is
21      End_Line : Boolean := True;
22    begin
23      Put (S1);
24      Put (S2);
25      if End_Line then
26        New_Line;
27      end if;
28    end Hg;
29
30    -----
31    -- Debug_Hg --
32    -----
33
34    procedure Debug_Hg (S : String) is
35    begin
36      if not Debug_On then
37        return;
38      end if;
39    end Debug_Hg;
40
41 end Screen_Output;

```

# Features

## Configurable and Extensible

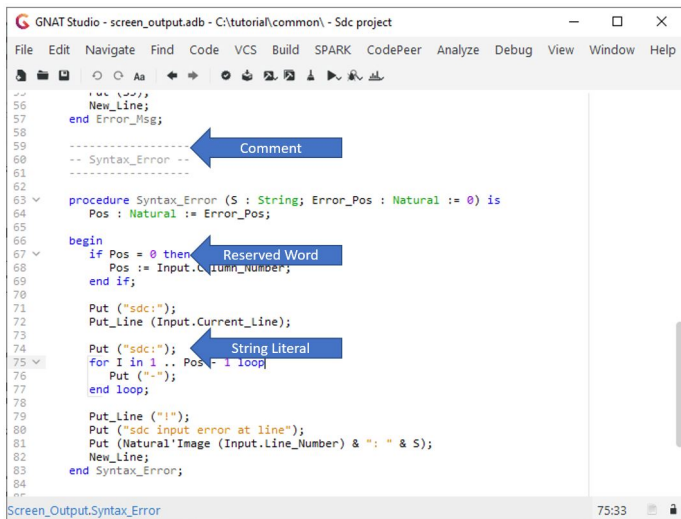
- Use your own color themes, favorite fonts, etc.
- Control layout for window panes within the application
- Create your own actions, with menu entries
  - Written in Python when appropriate
- Define your own editor text expansions (*aliases*)
  - Parameterized if necessary
- Define your own keyboard key assignments
  - E.g., binding a key sequence to an existing or user-defined action



## Provides Language-Sensitive Editing

- Syntax-directed coloring and highlighting
  - Statements, types, annotations, comments, etc.
- Indentation based on language syntax & surrounding code
- Automatic formatting as you type
  - Indentation, letter casing, coloring, etc.
- *Scope folding* to elide syntax-defined blocks of code
- Refactoring for entity renaming & subprogram extraction
- Semantics-based completion for both words and constructs

# Syntax Highlighting



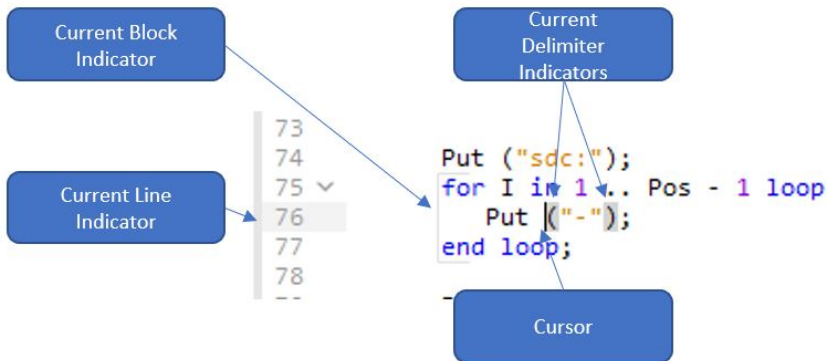
The screenshot shows the GNAT Studio interface with a code editor displaying Ada code. The code is color-coded to highlight different syntactic elements. Three blue callout boxes with arrows point to specific parts of the code:

- Comment:** Points to the line `-- Syntax_Error --` on line 60.
- Reserved Word:** Points to the keyword `then` on line 67.
- String Literal:** Points to the string `"sdc:"` on line 74.

```
56     New_Line;
57 end Error_Msg;
58
59 -----
60 -- Syntax_Error --
61 -----
62
63 procedure Syntax_Error (S : String; Error_Pos : Natural := 0) is
64     Pos : Natural := Error_Pos;
65
66 begin
67     if Pos = 0 then
68         Pos := Input.Current_Line;
69     end if;
70
71     Put ("sdc:");
72     Put_Line (Input.Current_Line);
73
74     Put ("sdc:");
75     for I in 1 .. Pos - 1 loop
76         Put ("-");
77     end loop;
78
79     Put_Line ("!");
80     Put ("sdc input error at line");
81     Put (Natural'Image (Input.Line_Number) & ": " & S);
82     New_Line;
83 end Syntax_Error;
```

Screen\_Output.Syntax\_Error 75:33

# Line / Block / Delimiter Highlighting



# Automatic Indentation

- Invoked when pressing enter key

- Modes

**None** No indentation performed

**Simple** Next line indented same as current line

**Extended** Based on language syntax & surrounding code

- Modes are controlled by preferences

■ **Edit** → **Preferences** → **Editor** → **Ada**

- Also invoked by pressing the indentation key

- Ctrl-Tab by default
- Can change via key manager

## Textual (Word) Completion

- Handy since source files often contain many references to the same words
- Invoked by Ctrl-/ after a partial word
  - Next possible completion will be inserted in the editor
  - Repeating cycles through list of candidate completions
- Candidates are those words occurring in the edited source file itself
- Key combination is customizable through the key manager dialog

# Smart (Semantic) Completions

- Completes the identifier prefix under the cursor
- Lists the results in a pop-up list
- Offers completions from the entire project
- Requires enabling *smart completion* preference
  - Hence computation of an entity database at GNAT STUDIO startup
- Allows configuring the time interval before pop-up
- Invocations
  - Automatically, on a partial word
  - Manually, by hitting control-space
  - Automatically, immediately after a dot
  - Automatically, immediately after an opening (left) parenthesis

# Smart Completions Example for Packages

```
procedure Error_Msg (S1 : String; S2 : String := "");
begin
  Put ("sdc error at line");
  Put (Natural'Image (Input.Line_Number) & ": ");
  Put (S1);
  Put (S2);
  Put (S3);
  New_Line;
  Ada.
end Err

-----
-- Synt
-----
tput.Erro
: $ Locat
  [
wind]
sdc.all
```

- Assertions
- Asynchronous\_Task\_Control
- Calendar
- Characters
- Command\_Line
- Containers
- Complex\_Text\_IO
- Decimal
- Directories
- Direct\_IO
- Dispatching

# Filtered Completion Proposals

```
procedure Error_Msg (S1 : String; S2 : S
begin
  Put ("sdc error at line");
  Put (Natural'Image (Input.Line_Number
  Put (S1);
  Put (S2);
  Put (S3);
  New_Line;
  Ada.T
end Err
```

- Tags
- ■ Task\_Attributes
- Synt ■ Task\_Identification
- ■ Task\_Initialization
- putput.Error ■ Task\_Termination
- es Locat ■ Text\_IO



# Information In Subprogram Proposals

```
procedure Pause is
begin
  Put ("Press a key to continue...");
  Skip_Line;
  Error
end
```

nd Sc

- Error
- Error\_Attributes
- Error\_Event
- Error\_In\_Regexp
- ▼ Error\_Type

Output

yes

```
..uuu :
:
:]
15-26
```

○ Error\_Msg

**Declaration:** [screen\\_output.adb:49](#)

**Parameters:**

```
S1 : in String
[S2 : in String := ""]
[S3 : in String := ""]
```

Prints the error message S1 followed by S2 followed by S3 on the screen.

# Formal Parameter Completions

```
procedure Pause is
begin
  Put ("Press a key to continue...");
  Skip_Line;
  Error_Msg (
end Pause;
```

```
end Screen_Output;
```

```
utput.Pause
```

```
es
```

```
|
```

```
end Screen_Output
```

```
] screen
```

```
5-26 10:04:37] pr
```

```
elapsed time: 0]
```

◆ params of Error\_Msg

● S1

● S2

● S3

● S2

Declaration: [screen\\_output.adb:49](#)

Type: String

Prints the error message S1 followed by S2 followed by S3 on the screen.

## Supports Source Navigation

- For Ada, C, and C++
- Hyperlinks allow project-wide traversal
  - Visiting declaration for a given name, the body of a routine, etc.
  - Including language-defined entities
- Contextual menus for navigating to current entity
- Dynamic dispatching calls are highlighted
- Traversable call graphs show entity relationships
  - E.g., "who calls this routine" or "who depends upon this package"
- "Tool-tips" pop up to show semantic information

# Outline view

Project

- ge
  - | config
  - | src
    - mage.ads
    - mage.adb
    - mage-apps.ads
    - mage-apps-simple\_loop.ads
    - mage-apps-simple\_loop.adb
    - mage-draw.ads

Scenario

Learn

Q- filter

▼ Ada.Numerics.Aux\_Long\_Long\_Float

- ▼ T
  - Q Acos (X : in T) return T
  - Q Asin (X : in T) return T
  - Q Atan (X : in T) return T
  - Q Cos (X : in T) return T
  - Q Cosh (X : in T) return T
  - Q Exp (X : in T) return T
  - Q Log (X : in T) return T
  - Q Pow (X : in T; Y : in T) return T
  - Q Sin (X : in T) return T
  - Q Sinh (X : in T) return T
  - Q Sqrt (X : in T) return T
  - Q Tan (X : in T) return T
  - Q Tanh (X : in T) return T

Outline

```

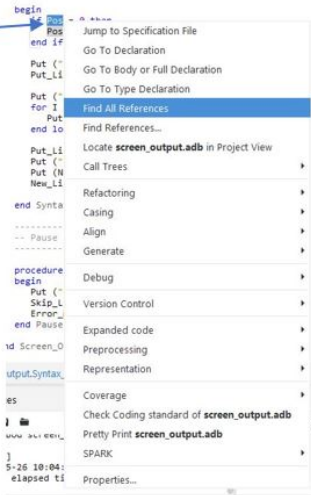
36 -- suitable implementation. It is thus qu
37 -- interfaces are suitable for cases in w
38 -- long double share the same representat
39
40 with Ada.Numerics.Aux_Linkers_Options;
41 pragma Warnings (Off, Ada.Numerics.Aux_Lin
42
43 package Ada.Numerics.Aux_Long_Long_Float is
44 pragma Pure;
45
46     subtype T is Long_Long_Float;
47
48     -- We import these functions as intrin
49     -- all as pure functions, because inde
50
51     function Sin (X : T) return T with
52       Import, Convention => Intrinsic, Exte
53
54     function Cos (X : T) return T with
55       Import, Convention => Intrinsic, Exte
56
57     function Tan (X : T) return T with
58       Import, Convention => Intrinsic, Exte
59
60     function Exp (X : T) return T with
61       Import, Convention => Intrinsic, Exte
62
63     function Sqrt (X : T) return T with
64       Import, Convention => Intrinsic, Exte
65
66     function Log (X : T) return T with
67       Import, Convention => Intrinsic, Exte
68
69     function Acos (X : T) return T with
70       Import, Convention => Intrinsic, Exte
71
72     function Asin (X : T) return T with
73       Import, Convention => Intrinsic, Exte
74
75     Ada.Numerics.Aux_Long_Long_Float.Exp

```

Locations   Call Trees   Messages   Tasks

# Editor's Contextual Navigation Menu

Cursor when  
right-clicked



# Tool-Tip Example

Cursor hovering over "Msg"

Subprogram Specification

```
procedure View is
begin
  for I in Tab'First .. Last loop
    Screen_Output.Msg (Values.To_String (Tab (I)));
  end loop;
  Screen_Output.Msg
end View;
end Stack;
```

```
procedure Msg (S1 : String; S2 : String := ""; End_Line : Boolean := True)
```

```
at screen_output.ads (5:4)
```

Prints message S1 followed by S2 on the screen. If End\_Line is True appends to the message a carriage return (ie it ends a line).

Subprogram Location

Subprogram Description  
(from comments after specification)

# Viewing Predefined and GNAT Source Files

The screenshot shows the GNAT Studio interface with the 'Help' menu open. The navigation path is: Help > GNAT Runtime > Standard > Ada > GNAT > Text\_IO. The 'Text\_IO' option is highlighted in blue. Below the menu, a snippet of Ada code is visible:

```

; : String; Error_Pos : Natural := 0) is
  _Pos;

1_Number;

```

On the right side of the image, a vertical list of predefined and GNAT source files is shown, each with a right-pointing arrow. The 'Text\_IO' entry is highlighted in blue. The list includes:

- Characters
- Command\_Line
- Containers
- Direct\_IO
- Directories
- Dispatching
- Exceptions
- Execution\_Time
- Interrupts
- Numerics
- Real\_Time
- Sequential\_IO
- Streams
- Strings
- Synchronous\_Task\_Control
- Tags
- Text\_IO**
- Wide\_Characters

# Call tree

```

58  -- define type Bodies_Array as an array of Body_Type indexed by
59  -- bodies enumeration
60  type Bodies_Array_T is array (Bodies_Enum_T) of Body_T;
61
62  procedure Move (Body_To_Move : in out Body_T; Bodies : Bodies_Array_T);
63
64  end Solar_System

```

Solar\_System.Move

Locations Call Tr

- Move\_All is called by
  - Private\_Types\_Ma
- Move\_All calls
  - Move
    - Compute\_X
      - Cos
        - Cos
        - Cos
        - Cos
        - Cos
      - Compute\_Y

Context menu actions:

- Go To Declaration
- Jump to Implementation File
- Go To Body or Full Declaration
- Go To Type Declaration
- Find All References
- Find References...
- Locate solar\_system.ads in Project View
- Call Trees**
  - Move is called by
  - Move calls
- Refactoring
- Casing
- Align
- Generate
- Debug
- Version Control
- Expanded code
- Preprocessing
- Representation
- GNATtest



# Running Applications

# Building Applications

- Uses multi-language builder `GPRBUILD` by default
  - Ada, C, C++, assembly, user-defined
- Supports any compiler callable on command line
  - Built-in support for GNAT, gcc, and make
- Provides easy navigation through error messages
- Provides automatic "code-fixing"
  - Manually invoked

```
80   Put ("sdc input error at line");
81   Put (Natural'Image (Input.Line_Number) & " : " & S);
82   New_Line;
83
84 end Syntax_Error;
85
86
87
88
89
```

Fix: missing " , "

## Integration with External Tools

- Common GUI for version control systems
  - Predefined support for many version control systems
  - Manual integration allowed for other tools
- GNAT-specific tools, if installed
  - GNAT SAS
  - SPARK
  - GNATTEST
  - GNATCOVERAGE
  - Etc.
- User-defined tools, with menu entries if needed

# Debugging Applications

# Symbolic Debugging

- Built in to GNAT STUDIO as a different "perspective"
  - Additional views, menu entries, and toolbar icons
- A graphical interface to GDB
- Uses a GDB enhanced to be Ada-aware
  - Task states, not just thread states
  - Advanced types<sup>1</sup> representations
- Same interface for native, cross, bare-board
  - Some targets may require target-specific setup
- Includes a GDB console for interactive commands

# Language Sensitive

- Multiple languages supported
  - Ada, C, C++ code in the same application
- Set variables, display expressions
  - Using language-specific syntax
- Browse source
  - Including language-defined entities

# Extensible

- You can call functions & procedures interactively
  - Using language-specific syntax
  - Very useful to print program specific info
  - No need to hardcode display routine calls within source
- Has powerful scripting facility
  - Can execute when app stops at a breakpoint
  - User defined commands (on the fly)
  - Command files (macros useful for your project)

# Fine-grained & Expressive Control

- Stepping
  - Over source line
  - Into and around subprograms
  - Over a single assembly instruction
- Breakpoints
  - Conditional & unconditional
  - Can execute a series of commands at breakpoint
- Viewable call stack
  - Move to any called routine on the call chain



## Exception Aware

- Halt when a *specific* exception is raised
- Halt when an *unhandled* exception is raised
- Halt when *any* exception is raised

# Tasking/Thread Aware

- View all tasks/threads in the application
- Set task specific breakpoints
- Switch among tasks by clicking on view entries

Debugger Tasks					
ID	TID	P-ID	Pri	State	Name
1	893960		15	Child Termination Wait	main_task
* 2	894710	1	15	Runnable	task_one
3	897d90	1	15	Runnable	task_two

# GNAT STUDIO Debug Perspective

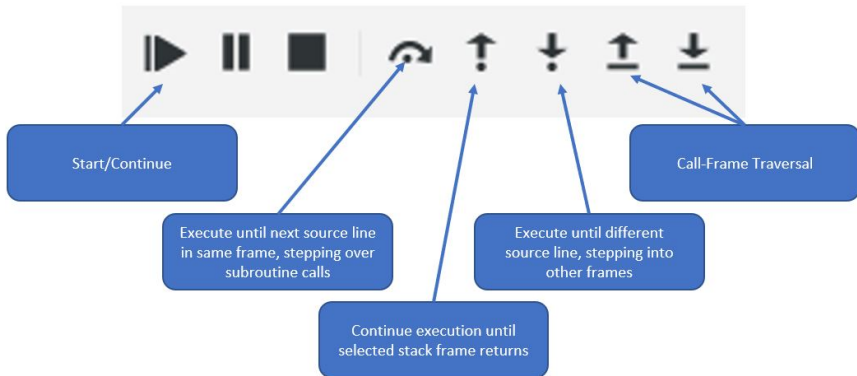
The screenshot displays the GNAT Studio Debugger Console interface for the 'Sdc project'. The main window shows the source code for 'screen\_output.adb', which includes a procedure 'Sdc' that reads a file and processes its contents. The code is as follows:

```
1  with Except;
2  with Screen_Output; use Screen_Output;
3  with Stack;
4  with Tokens; use Tokens;
5  with Ada.Text_IO; use Ada.Text_IO;
6  with Ada.Command_Line; use Ada.Command_Line;
7
8  procedure Sdc is
9     File : File_Type;
10
11 begin
12     Msg ("welcome to sdc. Go ahead type your commands ...");
13
14     if Argument_Count = 1 then
15         begin
16             Open (File, In_File, Argument (1));
17             exception
18                 when Use_Error | Name_Error =>
19                     Error_Msg ("Could not open input file, exiting.");
20                 return;
21             end;
22
23             Set_Input (File);
24         end if;
25
26         loop
27             -- Open a block to catch Stack Overflow and Underflow exceptions.
28
29             begin
30                 Process (Next);
31             end;
32             -- Read the next Token from the input and process it.
33
34             exception
35                 when Stack.Overflow =>
```

The interface includes a menu bar (File, Edit, Navigate, Find, Code, VCS, Build, SPARK, CodePeer, Analyze, Debug, View, Window, Help), a toolbar with various icons, and a sidebar with a Project Explorer showing the file structure. The bottom of the window features a Messages pane (Debugger Console) and a Locations pane. The Messages pane shows the following text:

```
File C:/tutorial/sdc.exe
Reading symbols from C:/tutorial/sdc.exe...
(gdb) |
```

# The Debugging Toolbar



# Data Window

- Displays values and their relationships in a table
- Each value is displayed in its own row
- Each row contains:
  - Name of the expression or variable
    - Components / elements can be expanded
  - Value
  - Type (Ada type definition)

# GNAT STUDIO Active In Debug Perspective

The screenshot displays the GNAT Studio interface in the Debug Perspective. The main window shows the source code of a program with a breakpoint set at line 23. The interface is annotated with four orange circles highlighting key debugging features:

- Variables:** A table on the left side of the editor showing the current state of variables.
- Breakpoints:** A table below the variables section listing active breakpoints.
- Call Stack:** A panel on the right showing the current call stack, with the active function highlighted.
- Debugger Console:** A panel at the bottom showing the output of the program, including thread information and the execution of a message procedure.

Name	Value	Type
file	0x0	ada.text_io.file
Debug_On	false	boolean
word		
i	<unknown>	4-byte float

Num	En	Type	Disp	File/Variable
1	✓	break	disable	main.adb
2	✓	break	disable	sd.cadb

```
9
10
11   Debug_On : constant Boolean := False;
12   -- when set, debugging information messages are output on the screen.
13
14   .....
15   -- Msg --
16
17   procedure Msg
18     (S1 : String;
19      S2 : String := "";
20      End_Line : Boolean := True)
21   is
22   begin
23     Put (S1);
24     Put (S2);
25     if End_Line then
26       New_Line;
27     end if;
28   end Msg;
29
30   -- Debug_Msg --
31
32   procedure Debug_Msg (S : String) is
```

Call Stack

- 0 screen\_output.msg
- 1 sdc

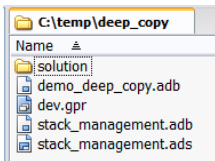
Debugger Console

```
gprb [New Thread 12204.0x154]
uil [New Thread 12204.0x3010]
d -
-P Thread 1 hit temporary breakpoint 3, sdc () at C:\tutorial\common\sd.cadb:9
C:\t
utor
ial\ 12 Msg ("Welcome to sdc. Go ahead type your commands ...");
sdc- [gdb] step
gpr screen_output.msg (s1=..., s2=..., end_line=true) at C:\tutorial\common\scree
```

## Workflow Example

# Starting GNAT STUDIO

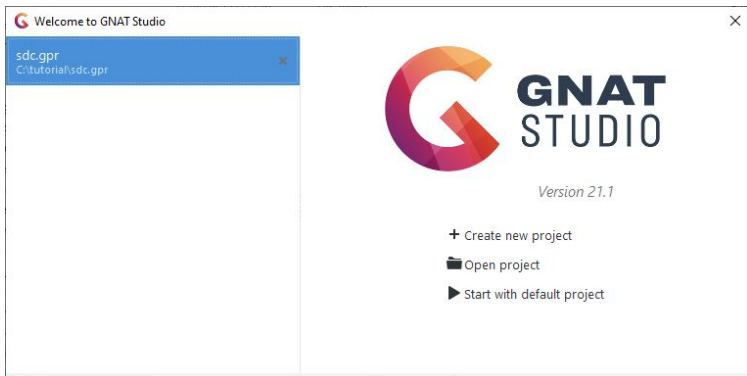
- From the desktop:
  - Double-click on the "project gpr" file icon in a file browser



- Or start GNAT STUDIO and use the Welcome Screen to select project
- From the command line:
  - Change to the directory containing the project file
  - Enter `gnatstudio` on the command line



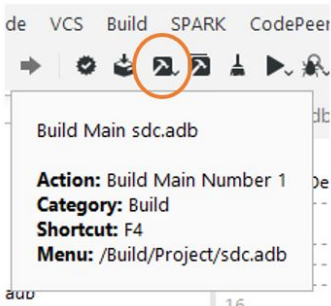
# GNAT STUDIO Welcome Screen



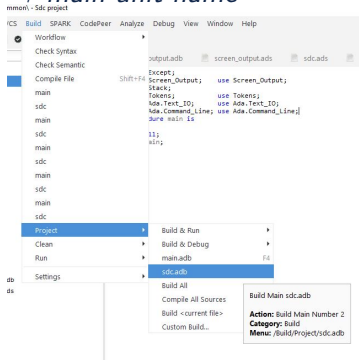
- Choose **Open Project**
  - Click **Browse** and go to your "dev" directory if the correct directory is not already indicated, or enter it directly
- Click **OK**

# Building Executables

- Press F4 (for first main in list)
- Or use "Build Main" icon

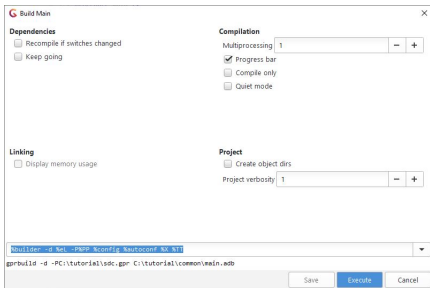


- Or click **Build** → **Project** → *main unit name*



# Chance To Change Build Switches

- May be displayed when build is invoked
- Just press OK



# Error In Source File and Locations View

The screenshot displays the GNAT Studio interface. The main editor window shows the source file `main.adb` with the following code:

```
1 with Except;  
2 with Screen_Output; use Screen_Output;  
3 with Stack;  
4 with Tokens; use Tokens;  
5 with Ada.Text_IO; use Ada.Text_IO;  
6 with Ada.Command_Line; use Ada.Command_Line;  
7 procedure main is  
8 begin  
9 null;  
10 end main;
```

The line `end main;` is highlighted in red, indicating an error. The status bar at the bottom of the editor shows `main` and the time `10:9`.

The Messages view at the bottom left shows the following output:

```
gprbuild -d -PC:\tutorial\sdc.gpr -XBUILD=DEBUG C:\tutorial\common\main.adb  
Compile  
[ada] main.adb  
main.adb:10:09: missing ";"  
gprbuild: *** compilation phase failed  
[2021-05-26 15:42:11] process exited with status 4, elapsed time: 00.83s
```

The Locations view at the bottom right shows the following structure:

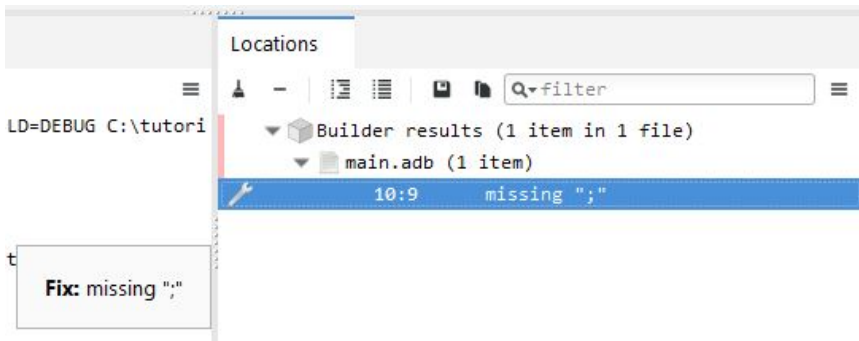
- Builder results (1 item in 1 file)
  - main.adb (1 item)
    - 10:9 missing ";"

## Results of Building

- Any error lines are displayed against a colored background in the source window
- *Locations* window displays error messages
- *Messages* window gives tool output results

## Using the Locations Window

- Can click on a line to go to that source location
- Click on the "wrench" icon to apply Code Fix



# Result of Code Fix via Wrench Icon

The screenshot displays the GNAT Studio interface. The main editor shows the source code for `main.adb`. Line 10, `end main;`, is highlighted in red, indicating a compilation error. Below the editor, the Messages pane shows the compilation output, and the Locations pane shows the error details.

```
1 with Except;
2 with Screen_Output; use Screen_Output;
3 with Stack;
4 with Tokens; use Tokens;
5 with Ada.Text_IO; use Ada.Text_IO;
6 with Ada.Command_Line; use Ada.Command_Line;
7 procedure main is
8 begin
9     null;
10 end main;
```

main 10:

Messages

```
gprbuild -d -PC:\tutorial\sdc.gpr -XBUILD=DEBUG C:\tutori
al\common\main.adb
Compile
[ada] main.adb
main.adb:10:09: missing ";"
gprbuild: *** compilation phase failed
[2021-05-26 15:42:11] process exited with status 4, elaps
```

Locations

Builder results (1 item in 1 file)

- main.adb (1 item)
- 10:9 missing ";"

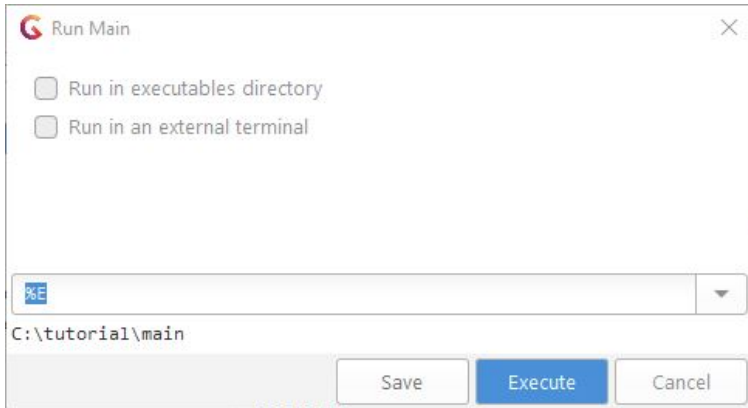
## Build the Executable After Fix

- Press F4 (for first main in list)
- Or use "Build Main" icon
- Or click **Build** → **Project** → *main unit name*

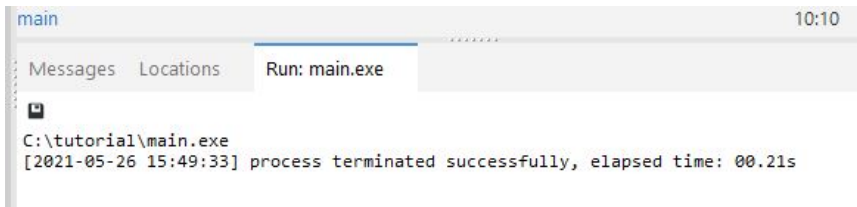


# Running The Program

- Click **Build** → **Run** → *main unit name*
- Leave *Use external terminal* unchecked
- Press **Execute**



# (Internal) Run Window

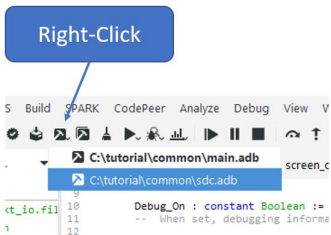


The screenshot shows the 'Run' window in GNAT Studio. The window title is 'main' and the time is '10:10'. There are three tabs: 'Messages', 'Locations', and 'Run: main.exe'. The 'Run: main.exe' tab is active and displays the following output:

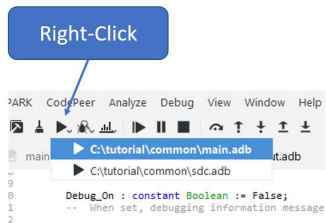
```
C:\tutorial\main.exe  
[2021-05-26 15:49:33] process terminated successfully, elapsed time: 00.21s
```

# When Multiple Mains Are Defined

## ■ Build Icon



## ■ Run Icon



# Help With GNAT STUDIO

The screenshot shows the GNAT Studio interface with the Help menu open. The menu items are: Welcome, Contents, GNAT Studio (highlighted), GNAT Runtime, GNAT, GPR, GNU Tools, XMLAda, Python, SPARK, CodePeer, GNATcoverage, and About. The 'GNAT Studio' item is expanded, showing a sub-menu with: Welcome, Tutorial, GNAT Studio User's Guide (highlighted), GNATdoc User's Guide, and Python extensions. A tooltip box is positioned over the 'GNAT Studio User's Guide' item, containing the following text:

Load the documentation for 'GNAT Studio User's Guide' into an external web browser

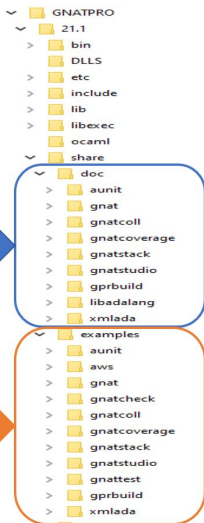
**Action:** display documentation GNAT Studio User's Guide  
**Category:**  
**Menu:** /Help/GNAT Studio/GNAT Studio User's Guide

In the background, a search bar is visible with the text 'Default search' and a search icon. A code editor window shows a snippet of code with a yellow highlight and the text 'ed till end of line.' below it.

# About GNAT STUDIO Help

- Information on GNAT STUDIO
  - Welcome (gets you to the Tutorial and the Users Guide)
  - Contents (which includes links to your reference manuals for GNAT, GDB and GCC, etc.)
- Information on other tools and capabilities
  - GNAT
  - GNAT SAS
  - GNU tools
  - GNAT Runtime
  - Python Extensions
- All GNATPro tools have a command-line argument `--help`

# User Guides and Examples



# Using Version Control Systems

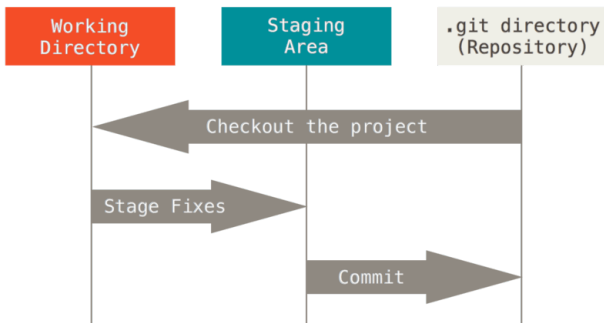
# What is version control

- System that records changes
  - to a file or set of files
  - over time
  - recall specific versions later
  - revert selected files back to a previous state
  - compare changes over time
  - who introduced an issue, and when
- GNAT Studio Supports many Version Control Systems (VCS)
  - Git
  - Subversion
  - CVS
  - Rational Clearcase
  - Mercurial

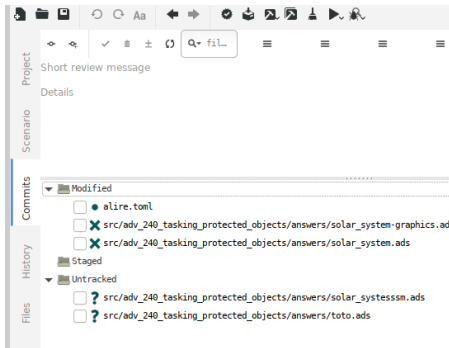


# What is Git

- A VCS
  - Used to demo the GNAT Studio VCS Features
- Distributed
  - No single database of reference
  - Most operations don't require a server
- Integrity checks
- 3 states



# GNAT Studio interface for Staging



- Tip: Renaming = Removing a file and creating a new file with the same content



# File Diff

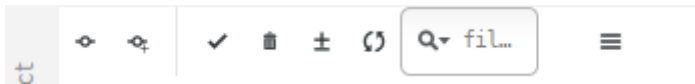
- Clicking on a file opens a diff

```

Diff alire.toml [HEAD]
1 ↓ diff --git alire.toml alire.toml
index 1464a66..da14271 100644
--- alire.toml
+++ alire.toml
@@ -1,15 +1,26 @@
-name = "labs_solar_system"
+authors = [
+  "Léo Germond",
+]
+description = "A set of SDL-based exercises to learn Ada"
+executables = [
+  "getting_started_main",
+]
+licenses = "GPL-3.0-or-later"
+maintainers = [
+  "Léo Germond <germond@adacore.com>",
+]
+maintainers-logins = [
+  "leogermond",
+]
-name = "labs_solar_system"
+tags = [
+  "training",
+  "labs",
+  "graphics",
+  "windowed",
+]
+version = "1.0.0"
+website = "https://public-training.adacore.com/doc/labs/solar_system/index.html"
-
+authors = ["Léo Germond"]
+maintainers = ["Léo Germond <germond@adacore.com>"]
+maintainers-logins = ["leogermond"]
-
-licenses = "GPL-3.0-or-later"
-executables = ["getting_started_main"]
-tag = ["training" "labs" "graphics" "windowed"]
diff --git alire.toml alire.toml

```

## Actions on the staging area



- Local
  - Undo local change(s)
  - Commit
  - Merge
- Distant
  - Push
  - Fetch
  - Pull = Fetch + Merge

# Commit a local change

The screenshot shows the commit dialog in GNAT Studio. The **Outline** section contains a **Short review message** field and a **Details** section. The **Commits** section shows a list of files: **Modified**, **Staged**, and **Untracked**. A red arrow points from the **Commit** button in the top toolbar to the **Short review message** field. A green arrow points from the **Details** section to the **Short review message** field. A blue arrow points from the **Commit** button to the **Modified** folder in the **Commits** section.

1. Select the staged change
2. Enter your message
3. Press "Commit"

# Lab

# GNAT Studio Lab

- Goals
  - Using GNAT STUDIO, you should be able to:
    - Build a project using existing source files
    - Fix coding issues by hand or automatically
    - Debug executables
  - Copy the two source directories ( `common` and `struct` ) to a work area

## Create Project - New Project

- Start GNAT STUDIO from the command line or the application menu
- In the Welcome dialog, select **Create new project**
  - Select **Simple Ada Project** and click **Next**
    - Fill in **Location** and **Settings** as appropriate
    - Click **Apply** to build the project



## Create Project - Project Settings

- Select **Edit** → **Project Properties...**
  - Navigate to the **Sources** → **Directories** tab
    - Remove the pre-populated directory
    - Add the **common** and **struct** directories
  - Navigate to the **Sources** → **Main** tab
    - Replace the **main.adb** file with **sdc.adb**
    - (Clicking the + icon brings up a list of all possible files)
  - Navigate to the **Build** → **Switches** → **Ada** tab
    - Select *Debug information* (so we can debug later)
    - Under *Warnings*, enable most warnings
  - Click **Save** to save settings

## Create Project - Build Project

- Press **F4** (and then **Execute**) to build the executable
  - There are errors in the supplied code!

# Error Fixing

- The error(s) appear in the **Locations** window
  - Clicking on the error line will jump to that line of code
  - For errors which GNAT STUDIO can fix, a wrench icon appears
    - In the **Locations** window
    - In the source file window
  - Clicking either of these wrenches should fix the problem
- Continue fixing errors (and warnings) until the executable builds

## Running the Executable

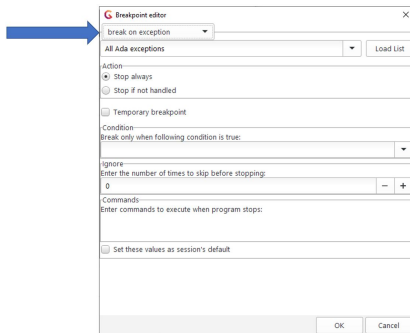
- This example is a simplistic postfix desktop calculator that accepts input from a file or interactively
  - For example, entering `1 2 + print` should give you the result 4, while `12 6 / print` will give you the result 2
- Run the executable via **Build** → **Run** or by pressing the right-pointing triangle icon
  - Enter `1 2 + print` as the command
  - **Internal Error** is not your fault - there is a bug in the code!

# Debugging the Executable

- **Internal Error** is printed when an exception is raised - let's try to find it
- Click the bug-like icon ( **Build & Debug** ) on the toolbar to start the debugger
- Click the **Continue** icon to start execution
  - Dialog has checkboxes - make sure *Stop at beginning of main subprogram* is checked so we can set a breakpoint
- Executable stops at main subprogram ( *Temporary breakpoint* )

## Debug - Setting an Exception Breakpoint

- We want to set a breakpoint when an exception is raised
- In the *Breakpoints* window, click the + icon
- Set the breakpoint type to *break on exception*
- Press **OK**
- Breakpoint appears in the *Breakpoints* window
- Click **Continue** to enter your data and see the exception



## Debug - Following an Breakpoint

- Execution stops where exception is raised
  - Not always in your actual code
  - In **Debugger Console** exception information is presented
  - In **Call Stack** window, you can see where you are in the call stack
    - Click on the first entry that looks like your code
  - To see current value of an object, hover over it
    - To track the value, right-click and select **Debug** →  
**Display <> in Variables view**