



Ada Development Environment on Linux



Sowebio SARL
15, rue du Temple
17310 – St Pierre d'Oléron – France

Capital 15 000 EUR – SIRET 844 060 046 00019 – RCS La Rochelle – APE 6201Z – TVA FR00844060046

Ada Development Environment on Linux

Ed.	Release	Comments	
1	20230527	Initial release.	sr
6	20230530	Add GNATStudio settings	sr
8	20230530	Add section Convert a project to Alire	sr
10	20230604	Add GNATStudio documentation links	sr
17	20230605	Move AIDE documentation to this document	sr
32	20230609	Add section Convert a library to Alire, change Convert a project to Alire	alf
34	20230728	Unified logos	sr
36	20231128	Various updates in GNAT Toolchain chapter, relating to GCC 13.2	sr
36			

- Authors

Stéphane Rivière [Number Six] - stef@genesix.org [CTO Sowebio]

- Acknowledgments

Adacore GNAT team, Alire author Alejandro Mosteo.

- Manual

Stéphane Rivière [Number Six] - stef@genesix.org [CTO Sowebio - FM1US/F1USA¹]

The “Excuse me I’m French” speech - The main author of this manual is a Frenchman with basic English skills. Frenchmen are essentially famous as frog eaters². They have recently discovered that others ~~forms of communication~~ languages are widely used on earth. So, as a frog eater, I’ve tried to write some stuff in this foreign dialect loosely known here under the name of English. However, it’s a well known fact that frogs don’t really speak English. So your help is welcome to correct this bloody manual, for the sake of the wildebeests, and penguins too.

- Syntax notation

Inside a command line:

- A parameter between brackets [] is optional;
- Two parameters separated by | are mutually exclusives.

An important notice:

⇒ This is an important notice !

- Edition

1 36 - 2023-11-30

¹International amateur radio call sign - https://en.wikipedia.org/wiki/Amateur_radio.

²We could be famous as designers of the Concorde, Ariane rockets, Airbus planes or even Ada computer language but, definitely, Frenchmen have to wear beret with bread baguette under their arm to go eating frogs in a smokey tavern. That’s *le cliché* :]

<https://this-page-intentionally-left-blank.org>



Contents

Introduction.....	9
1 About this manual.....	9
2 Syntax notation.....	9
3 About the Ada Community.....	9
3.1 Inspiration, ideas, help and more.....	9
4 Manual background.....	10
5 About Ada.....	11
5.1 Introduction.....	11
5.2 Why use Ada.....	11
5.3 The ending word.....	12
GNAT toolchain.....	13
1 GNAT Studio.....	13
1.1 Documentation.....	13
1.2 Install.....	13
1.2.1 24.0w release.....	13
1.2.2 23.0w release.....	14
1.3 Setup.....	14
1.3.1 General.....	14
1.3.2 Editor.....	15
1.3.3 External commands.....	15
1.3.4 Windows.....	15
1.3.5 Build targets.....	16
1.3.6 Plugins.....	16
1.3.7 Shortcuts.....	16
2 Alire repository manager.....	17
2.1 Links.....	17
2.2 Install.....	17
2.3 Use.....	18
2.3.1 List all crates.....	18
2.3.2 Search for some specific packages.....	18
3 Native Linux compiler.....	18
3.1 Create Alire repository.....	18
3.2 Installing Gprbuild.....	18
3.3 List all GNAT packages.....	18
3.4 Installing native compiler.....	18
3.5 Select toolchain.....	19
3.6 Other components.....	19
4 Gpb utility.....	19
4.1 Login automation.....	19
4.2 Spaces in paths.....	19
4.3 Build string.....	20

4.4	Example of build with SCP.....	20
4.5	Open the project.....	21
4.6	Build Gpb.....	21
4.7	Installation.....	21
4.8	Host authenticity with SCP.....	21
5	Documentation.....	21
	GNAT Studio full workflow.....	23
1	Project example Hello from Alire repository.....	23
1.1	Get Hello.....	23
1.2	GNAT Studio IDE.....	23
1.2.1	Clean all.....	24
1.2.2	Build all.....	24
2	Create native Alire project.....	25
2.1	Create Alire project.....	25
2.2	GNAT Studio IDE.....	25
2.2.1	Build all.....	25
2.3	Run.....	26
3	Convert a project to Alire.....	27
3.1	Alire Kalle repository conversion.....	27
3.2	Alire Kalle repository customization.....	28
3.3	Kalle build.....	28
3.4	Kalle tests programs.....	29
4	Convert a library to Alire.....	30
4.1	Alire repository conversion.....	30
4.2	Setup dependencies with Alire.....	30
4.3	Edit and build Alire library.....	31
4.4	Test library.....	31
	Learning Ada.....	33
1	Introduction.....	33
2	Requirements.....	33
3	Historical books.....	33
4	Ada books.....	33
5	Ada courses.....	34
6	Ada links.....	34
	Coding examples.....	35
1	HAC Ada interpreter.....	35
2	GNATStudio Examples.....	35
2.1	Drink dispenser.....	35
2.2	Sorting algorithm.....	35
3	GNATStudio Examples [AVR 8 bits microcontroller].....	35
4	Programs from the MX Team.....	35
4.1	Mx.....	36
4.1.1	Overview.....	36
4.1.2	Build.....	36
4.1.3	Usage.....	36
4.2	Visual.....	36

4.2.1	Overview.....	36
4.2.2	Build.....	36
4.2.3	Usage.....	37
4.3	Updates from original 2004 release.....	37
4.3.1	Overview.....	37
Programming basics.....		39
1	Tools.....	39
2	Analysis.....	40
2.1	Methods overview.....	40
2.2	Top-Down example.....	40
2.2.1	Problem's decomposition.....	41
2.2.2	Pseudo-code.....	42
3	Modular and structured programming method.....	43
3.1	Introduction.....	43
3.2	Program Structure Diagram.....	44
3.2.1	Process detailed.....	44
3.3	Pseudo-code.....	45
3.3.1	Main module.....	45
3.3.2	Other modules.....	45
3.3.3	Sequence.....	46
3.3.4	Module call.....	46
3.3.5	If... else... end if.....	46
	If... elsif... else... endif.....	47
3.3.6	Case... when... else... end case.....	47
3.3.7	Do while... end do.....	48
3.3.8	Loop... until.....	48
3.4	Functions.....	49
4	Boole algebra.....	49
4.1	Identities, properties and De Morgan's laws.....	49
4.1.1	Identities.....	49
4.1.2	Properties.....	49
4.1.3	De Morgan's law.....	50
4.2	Practical advises.....	50
5	Basics algorithms.....	50
5.1	Initial reading & current reading in loops.....	50
FAQ.....		51
1	Issues & solutions.....	51
1.1	Error when trying to reading documentation: No HTML browser specified....	51
1.2	No GNATStudio icon in dock.....	51
1.3	Association lost between .gpr project files and GNATStudio.....	51
1.4	GNAT Runtime help tree is altered in GNATStudio.....	52
1.5	How file association is processed by the system.....	53
1.6	Where are stored GNATStudio configuration files ?.....	54
2	Ada.....	54
2.1	Check calls to external libraries.....	54
2.2	Library integration with .gpr.....	54

2.3	Program calls analysis.....	55
2.4	Statically link an external library to an executable.....	55
2.5	Statically linked executable embedding the run-time system.....	56
Appendices.....		61
1	Copyrights & credits.....	61
1.1	Library Licence.....	61
1.1.1	GPL v3 compatibility with others licenses.....	61
1.2	Manual license.....	61
2	To-do list Documentation.....	61
3	To-do list Software.....	61
3.1	Erroneous message after exception handling.....	61
4	Links.....	61
4.1	Ada.....	61
4.2	Others.....	61
4.3	People.....	62
4.3.1	Ludovic Brenta.....	62
4.3.2	Stéphane Carrez.....	62
4.3.3	Frédéric Praca.....	62
4.3.4	Gautier de Montmollin.....	62

Introduction

Keep It Simple, Stupid.
Clarence Leonard "Kelly" Johnson



1 About this manual

For a development workstation under GNU/Linux, the system of choice is probably Ubuntu or one of its derivatives.

The author is not an Ada expert, nor a well-informed GNU free tools user, nor a fluent english writer : suggestions in order to improve this manual are very welcome.

2 Syntax notation

Inside a command line :

- A parameter between brackets [] is optional ;
- Two parameters separated by | are mutually exclusives.

3 About the Ada Community

At first, thanks to the Ada Community, definitely one of the best.

3.1 Inspiration, ideas, help and more

AdaCore Ada compiler - <https://www.adacore.com/community>

Rolf Ebert - <https://github.com/RREE>

Daniel Feneuille - <http://d.feneuille.free.fr>

Gautier de Montmollin - <https://github.com/zertovitch>

Pascal Pignard - <https://github.com/Blady-Com>

Jean-Pierre Rosen - <https://adalog.fr>

David Sauvage - <https://www.adalabs.com>

Special thanks to Ada gurus Daniel Feneuille, Gautier de Montmollin and Jean-Pierre Rosen. The chapter heading quotes are extracted from Murphy's Law and other reasons why things go wrong - A. Bloch. They come from <https://www.adalog.fr> site created by Jean-Pierre Rosen.

4 Manual background

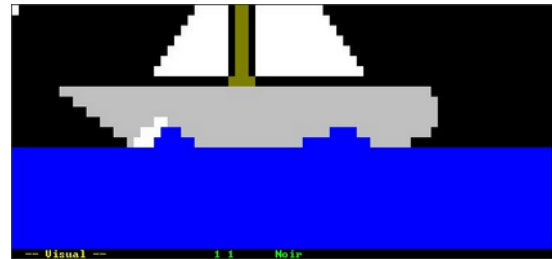
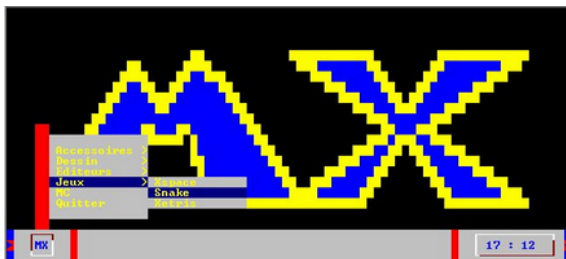
This manual has its roots from AIDE³ 0.5 [2002] to AIDE v1.4 [2005], with an edition for Windows that was favored by the 5th edition of the LSM [Libre Software Meeting] on Bordeaux in 2004 the 8th of July. After introducing AIDE, Martin and Xavier [13 years both at this time] has explained how they learn programming in Ada.

Let's hear from Ludovic Brenta⁴, a prominent and well-known member of the Ada community:

"I was most impressed by two 13-year-old youths who started learning programming in February this year, and are already Ada die-hard after playing with Python for a while, and also looking at Lisp, C and Java".

They understand that Ada is not a fashionable language but still prefer using a good language than a fashionable one. Even more stunning, they even prefer using Emacs instead of more graphical IDEs such as GPS⁵! They've written a 2000-line text-mode application in Ada that allows them to draw pictures using ASCII block characters, save them into text files, read back and display them. They designed the file format themselves, and it turns out it is quite similar to XPM.

They have a second application that uses these files to display a "Start" menu with a number of applets, one of which is a fully working calculator. The father of one of these youths, Stéphane Rivière of AIDE fame, taught them the basics of Ada during 45-minute courses on Sundays, and they did all the rest by themselves with very little supervision. After only 4 months since their first exposure to programming, they understand and routinely use separate compilation and encapsulation, and were asking me questions about multitasking and game programming in Ada!"



During these years, AIDE was a tool of choice for Ada trainers. They could set up an Ada training room in minutes on any PC!

Then time passed, Windows no longer exists for us, nor does it seem relevant for a free software developer concerned with his tools. Martin and Xavier had dreamed of a version of AIDE for Debian. It was time, in 2019, to re-create AIDE for our own needs - high availability servers cluster management and web applications - and to share it with the free software community.

³ Ada Instant Environment Development, a ready to use Ada environment for Windows with Unix tools.

⁴ <https://comp.lang.ada.narkive.com/aKzBkWD5/ann-ada-on-the-2004-libre-software-meeting>

⁵ Previous name of GNATStudio, GPS was renamed in 2020.

Some time later, Alire, the Ada package manager, came along, making AIDE obsolete. Once again, we revised this manual to keep it up to date with the latest trends in Ada development. Alire is an important event for a more effective use of the Ada language

5 About Ada

Some general thoughts about Ada.

5.1 Introduction

This language is not known enough yet, at least not to the majority of us, much to the detriment of many potential users for that matter. Compared to the fashionable languages, Ada is more portable, more readable, allows for higher abstraction levels and has features and functionalities unseen in other languages. Ada also allows a more comfortable experience in system programming⁶ and proves itself light enough to be usable on low class 8 bit processors⁷.

Ada is the name of the first programmer to ever exist in humanity. And this first programmer was a woman: Augusta Ada Byron King, Countess of Lovelace, born in 1815, daughter of Byron, the great poet, Charles Babbage's assistant, she wrote programs destined to run on his famous machine.

Ada is an American military norm⁸ as well as an international civil norm⁹, it is the first object oriented language to be standardized at an international level. All Ada compilers must strictly adhere to the standard. There are hundreds of compilers destined to run on that many platforms but all of them will produce a code that runs identically.

Ada is used everywhere security is critical: Airbus [A320, A330, A340, A350, A380 civil airplanes and A400 military airplane], Alstom [High speed train], Boeing [777 and 787 airplanes], EADS [Eurofighter, Ariane rockets, ATV spacecraft, all European spaces probes], Dassault [Rafale fighter], Lockheed Martin [F22 Raptor], STS [line 14 Meteor Paris unmanned metro system], NASA [Electric power supply of the International Space Station, European service module for Orion spacecraft Moon missions].

The list goes on and on. Everywhere reliability and security must come first, Ada is the language of choice.

5.2 Why use Ada

Ada was created because software engineering is a human activity. Humans make mistakes, the Ada compiler is friend to developers. Ada is also friend to project managers for large scale development. An Ada application is written, expanded and maintained very naturally. For these reasons, Ada is also friend to executives. Ada is the language of happy programmers, managers and users.

⁶Thanks to it's representation clauses that obliterates the need to use bit masking for XORed for bit manipulation. This functionality *essential to system programming* is simply not there in pure C or even in Assembly language.

⁷Components that have at their disposal a couple dozen bytes of RAM and a couple Kilobytes of programming memory.

⁸MIL-STD-1815

⁹ISO/IEC 8652



Because Ada is a comfortable language by it's expressiveness and a restful language by it's reliability, humans involved with Ada also reflect the image of their language. The Ada community is a very comfortable community to visit and most meetings are very enlightening. Free libraries are numerous and are usually of a very high quality. Finally, the Ada community is very highly active and by now growing again.

5.3 The ending word

When Boeing decided, two decades ago, that all software for the 777¹⁰ would be exclusively written in Ada, the corporate associates of the constructor made the remark that they were using, for a long time, languages such as C, C++ and assembly language and that they were fully satisfied with them. Boeing simply answered that only firms that could provide Ada software would be considered in contracts offerings. Therefore, the firms converted themselves to Ada.

Today, the development of software for the Boeing 777 nicknamed « The Ada Plane », has been performed and it is essentially thanks to the very big commercial success of this plane that Boeing was able to maintain the revenues created by its civil activities¹¹.

And what do the Boeing partner firms do from now on ? They continue to develop their new software in none other than... Ada, and here's why:

- They noticed that the length of time to convert developers to Ada is usually rather short. In a week, the developer is comfortable enough to write software in Ada and in less than a month, he feels totally comfortable with the language;
- These firms did their accounting: written in Ada, software costs less, present less anomalies, are ready sooner and are easier to maintain.

¹⁰ The Boeing 777 is the world's biggest two engines plane and the first civil Boeing having electrical flight commands, ten years later the Airbus A320.

¹¹ This text was written well before the tragic engineering failure of the 737 Max.

GNAT toolchain

Doubling the number of programmers on a late project does not make anything else than double the delay.
Second Brook's Law



1 GNAT Studio

1.1 Documentation

Tutorial: https://docs.adacore.com/live/wave/gps/html/gps_tutorial/index.html

Documentation: https://docs.adacore.com/live/wave/gps/html/gps_ug/index.html

Release notes: https://docs.adacore.com/live/wave/gnatstudio-release-notes/html/gnatstudio_release_notes/index.html

1.2 Install

Pick the latest in <https://github.com/AdaCore/gnatstudio/releases/latest>.

1.2.1 24.0w release

AppImage to /opt:

```
user@system : wget https://github.com/AdaCore/gnatstudio/releases/download/gnatstudio-cr-20230501/
GNAT_Studio-x86_64.AppImage
user@system : chmod +x GNAT_Studio-x86_64.AppImage
user@system : ./GNAT_Studio-x86_64.AppImage --appimage-extract
user@system : cd ./squashfs-root/usr
user@system : sudo ./doinstall
Default install : /opt/gnatstudio
user@system : PATH="/opt/gnatstudio/bin:$PATH"; export PATH
user@system : echo >> ~/.bashrc; echo 'export PATH=/opt/gnatstudio/bin:$PATH' >> ~/.bashrc
```

➤ At the very end of \$HOME/.bashrc add export PATH=/opt/gnatstudio/bin:\$PATH

Create \$HOME/.local/share/applications/gnatstudio.desktop :

[Desktop Entry]

Ada Development Environment on Linux

```
Name=GnatStudio
Icon=/opt/gnatstudio/share/gnatstudio/icons/hicolor/32x32/apps/gnatstudio_logo.png
Exec=/opt/gnatstudio/bin/gnatstudio
Terminal=false
Type=Application
MimeType=application/x-adagpr
Categories=Development;
StartupWMClass=gnatstudio_exe
```

1.2.2 23.0w release

Linux-bin to \$HOME/opt:

```
user@system : mkdir $HOME/opt ; cd $HOME/opt

user@system : wget https://github.com/AdaCore/gnatstudio/releases/download/gnatstudio-cr-20220512/gnat-
studio-23.0w-20220512-x86_64-linux-bin.tar.gz

user@system : tar -xvf gnatstudio-23.0w-20220512-x86_64-linux-bin.tar.gz

user@system : cd gnatstudio-23.0w-20220512-x86_64-linux-bin

user@system : sudo ./doinstall

Default install : /opt/gnatstudio

user@system : PATH="/opt/gnatstudio/bin:$PATH"; export PATH

user@system : echo >> ~/.bashrc; echo 'export PATH=/opt/gnatstudio/bin:$PATH' >> ~/.bashrc
```

➤ At the very end of \$HOME/.bashrc add export PATH=/opt/gnatstudio/bin:\$PATH

Create \$HOME/.local/share/applications/gnatstudio.desktop :

```
[Desktop Entry]
Name=GnatStudio
Icon=/home/sr/opt/gnatstudio/share/gnatstudio/icons/hicolor/32x32/apps/gnatstudio_logo.png
Exec=/home/sr/opt/gnatstudio/bin/gnatstudio
Terminal=false
Type=Application
MimeType=application/x-adagpr
Categories=Development;
StartupWMClass=gnatstudio_exe
```

1.3 Setup

Launch GNATStudio

The very first time, a configuration wizard is displayed. Set the color theme of your choice and click on [Skip & Use Defaults] at the upper right window corner.

➤ Only the relevant commands are mentioned, whether they are left at their default value or not.

Menu > Edit > Preferences...

1.3.1 General

- Main

```
Behavior
[x] Auto save (default)
[x] Save desktop on exit (default)
```

Default Builder
(o) Gprbuild (default)

Charsets
Character set: Unicode UTF-8¹² (instead of Western/Latin-1 (ISO-8859-1))

Clipboard
Clipboard size: 50 (instead of 10)

- Custom styles

Theme: Adwaita (default)
Default font: DejaVu Sans 9 (default)
Monospace font: DejaVu Sans Mono 8 (default)
Command window background: white (default)
Toolbar style: Small Icons (default)

- Key Shortcuts

Build > Build All > [Add] > F9 > [Remove]
Editor > Center Line > [Add] > Alt + C
Editor > Comment lines Ctrl + / > [Remove] > [Add] > Ctrl + Shift + >
Editor > Delete line > [Add] > Ctrl + Y > [Remove]
Editor > Subprogram box > [Add] > F10
Editor > Uncomment lines Ctrl + ? > [Remove] > [Add] > Ctrl + < [Remove]

1.3.2 Editor

- Ada

(o) Simple indentation (instead of extended)
[] Indent comments (instead of [x])
It should be wised to not change other options.

1.3.3 External commands

- General

List processes: sh -c ""(ps x 2> /dev/null || ps -u \[extract_itex]USER 2> /dev/null || ps) | cat"" (default)
Execute command: xterm -hold -e (default)
Print command: a2ps (default)

You may find useful to hardcode your browser path if GNATStudio can't find it:
HTML browser: /usr/bin/firefox %u

1.3.4 Windows

Floating Windows
You may prefer to use GNATStudio with floating windows:
[] or [x] All floating

Notebook Tabs
You may find this settings useful using a large screen:
Notebook tabs position: Right
Notebook tabs position: Horizontal

¹²GNATStudio uses Unicode internally

1.3.5 Build targets

A setting page of interest.

1.3.6 Plugins

You may wish to add theses plugins:

```
To be used with -bargs -E switch
[x] Addr2line
[x] Auto Locate File
[x] Build and run all
[x] Copy Paste
[x] Copy Paste Toolbar
[x] Cov Export
Important for your comfort
[x] Enter
Mandatory if you want to respect the Ada RTS Style
[x] Highlight Column with margin Column at 80
Depending of your choice but highly recommended
[x] Prevent Project Edition
[x] Separate
[x] Treemove
```

1.3.7 Shortcuts

- Comment box for subprograms

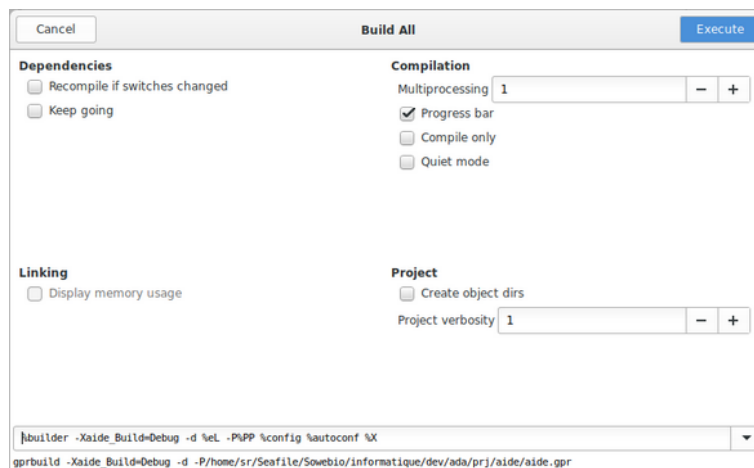
[F10] will generate a comment box with the same name above the subprogram declaration:

```
-----
-- Process_A_File --
-----

procedure Process_A_File (TXT_Name: String) is
-- Process a bank statement
```

- Build all

[F9] triggers the build all window:



Default command line [at the window bottom]: %builder -d %eL -P%PP %config %autoconf %X

- Comment or comment a block
[Ctrl] + [Shift] + [>] Comment the selected block.
[Ctrl] + [<] Uncomment the selected block.
- Debug - Step
[F5]
- Debug - Step out
[F6] Execute the program until the next source line stepping over subprograms calls
- Debug - Finish
[F7] Continue execution until selected stack frame returns
- Debug - Run
[F8] Continue execution until next breakpoint
- Delete a line
[Ctrl] + [Y] Remember Wordstar¹³

2 Alire repository manager

Alire is the Ada Library REpository manager for the Libre and Open Source Ada ecosystem.

A comprehensive presentation paper, from the Alire author, can be found in AUJ Vol 39, Number 3, Sept 2018, P 189. <http://www.ada-europe.org/archive/auj/auj-39-3.pdf>

2.1 Links

<https://github.com/alire-project>
<https://alire.ada.dev>
<https://alire.ada.dev/docs>
<https://gitter.im/ada-lang/Alire>
<https://www.reddit.com/r/ada>
<https://twitter.com/mosteobotic>

2.2 Install

Pick the latest at <https://github.com/alire-project/alire/releases>

```
user@system : wget https://github.com/alire-project/alire/releases/download/v1.2.2/alr-1.2.2-bin-x86_64-linux.zip
```

```
user@system : sudo unzip -j -o alr-1.2.2-bin-x86_64-linux.zip bin/alr -d /usr/local/bin
```

```
Archive: alr-1.2.2-bin-x86_64-linux.zip
  inflating: /usr/local/bin/alr
```

¹³ <https://en.wikipedia.org/wiki/WordStar>

2.3 Use

2.3.1 List all crates

```
user@system : alr search --list
```

2.3.2 Search for some specific packages

```
user@system : alr search --full --external-detect avr (all versions)
```

```
user@system : alr search --external-detect avr (last versions)
```

3 Native Linux compiler

3.1 Create Alire repository

```
user@system : mkdir --parents $HOME/opt/alire; cd $HOME/opt/alire
```

3.2 Installing Gprbuild In \$HOME/opt/alire:

```
user@system : alr get gprbuild
```

3.3 List all GNAT packages

```
user@system : alr search --external-detect gnat
```

gnat_arm_elf	13.2.1	The GNAT Ada compiler - ARM cross-compiler
gnat_avr_elf	13.2.1	The GNAT Ada compiler - RISC-V cross-compiler
gnat_math_extensions	1.1.0	Eigenvalues, eigenvectors for matrices
gnat_native	13.2.1	The GNAT Ada compiler - Native
gnat_riscv64_elf	13.2.1	The GNAT Ada compiler - AVR cross-compiler
gnatcoll	24.0.0	GNAT Components Collection - Core packages
gnatcoll_gmp	24.0.0	GNAT Components Collection - GNU Mult. Precision Arithmetic
gnatcoll_iconv	24.0.0	GNAT Components Collection - iconv binding
gnatcoll_lzma	24.0.0	GNAT Components Collection - lzma binding
gnatcoll_omp	24.0.0	GNAT Components Collection - OpenMP binding
gnatcoll_postgres	24.0.0	GNAT Components Collection - postgres
gnatcoll_python	21.0.0	GNAT Components Collection - python2 binding
gnatcoll_python3	23.0.0	GNAT Components Collection - python3 binding
gnatcoll_readline	24.0.0	GNAT Components Collection - readline binding
gnatcoll_sql	24.0.0	GNAT Components Collection - sql
gnatcoll_sqlite	24.0.0	GNAT Components Collection - sqlite
gnatcoll_syslog	24.0.0	GNAT Components Collection - syslog binding
gnatcoll_xref	24.0.0	GNAT Components Collection - xref
gnatcoll_zlib	24.0.0	GNAT Components Collection - zlib binding
gnatcov	22.0.1	Coverage Analysis Tool
gnatdist_garlic	6.0.1	The configuration tool gnatdist for GARLIC
gnatdoc	23.0.0	GNAT Documentation Generation Tool (as `gnatdoc4` binary)
gnatprove	13.2.1	Automatic formal verification of SPARK code
gpr_unit_provider	23.0.0	GNAT Project File Library

3.4 Installing native compiler

```
user@system : alr get gnat_native
```

3.5 Select toolchain

```
user@system : alr toolchain --select

Please select the gnat version for use with this configuration
 1. gnat_native=13.2.1
 2. None
 3. gnat_arm_elf=13.2.1
 4. gnat_avr_elf=13.2.1
 5. gnat_riscv64_elf=13.2.1
 6. gnat_arm_elf=13.1.0
 7. gnat_avr_elf=13.1.0
 8. gnat_native=13.1.0
 9. gnat_riscv64_elf=13.1.0
10. gnat_arm_elf=12.2.1
 a. (See more choices...)
Enter your choice index (first is default):
> 1

user@system : alr toolchain

CRATE      VERSION      STATUS      NOTES
gprbuild   22.0.1       Default
gprbuild   2021.0.0+0778 Available Provided by system package: gprbuild
gnat_native 13.2.1       Default
```

3.6 Other components

Depending on your projects, you may also want to install these crates :

```
user@system : alr get gnatcov
user@system : alr get gnatdoc
user@system : alr get gnatprove
user@system : alr get gnatcoll
```

4 Gpb utility

✦ Gpb is very useful when developing locally but testing on a remote server.

GPB Utility is available at: <https://github.com/sowebio/aide-gpb>

Gpb¹⁴ is a Gprbuild stub to handle distant targets. It allows a network copy, through SCP, of the current binary project after build, with bell[s] ;)

✦ The real Gprbuild is renamed gprbuild_org.

If the -XGpb_Scp= parameter is set, Gpb copies the binary accordingly to the SCP destination path. A second sound will be emitted after the end of the SCP copy [useful with slow links].

4.1 Login automation

A valid SSH key for login automation is mandatory.

4.2 Spaces in paths

✦ /\ The path to the gpr project file must not contain spaces /\

¹⁴ Gpb uses Gnatgpr from Adalabs <https://www.adalabs.com>. Gpb was also inspired by Adalab's director, David Sauvage. Many thanks to him.

```
/home/sr/Seafire/Sowebio/testdev/github/gpb/gpb.gpr => right
/home/sr/Seafire/Sowebio/test dev/github/gpb/gpb.gpr => wrong
```

❖ /!\ The path to the project itself must not contain spaces either /!\

4.3 Build string

This string:

```
%builder -Xaide_Build=Debug -XGpb_Scp=root@host.domain.tld:/usr/local/bin -d %eL -P%PP %config %autoconf
%X -s
```

Is translated as follows:

```
gprbuild -Xaide_Build=Debug -XGpb_Scp=root@host.domain.tld:/usr/local/bin -d -P/home/sr/Seafire/
Sowebio/informatique/github/gpb/gpb.gpr -s
```

Extra options:

```
-XGpb_Beep=off|ansi|[bell] ansi=console beep, bell=neat bell through pulseaudio (default)
```

4.4 Example of build with SCP

Local build, then SCP copy to remote server:

GnatStudio Build All (with modified F9 shortcut)

```
%builder -XGpb_Build=Debug -XGpb_Scp=root@domain.tld:/usr/local/bin
-d %eL -P%PP %config %autoconf %X -s
```

```
gprbuild -XGpb_Build=Debug -XGpb_Scp=root@domain.tld:/usr/local/bin
-d -P/home/sr/Seafire/Sowebio/informatique/github/gpb/gpb.gpr -s
```

Build trace:

```
gprbuild -XGpb_Build=Debug -XGpb_Scp=root@domain.tld:/usr/local/bin
-d -P/home/sr/Seafire/Sowebio/informatique/github/gpb/gpb.gpr
-Xgpb_Build=Debug -s
```

```
Gprbuild stub for GnatStudio - gprbuild v0.1
Copyright (C) Sowebio SARL 2020-2021, according to GPLv3.
```

```
Gprbuild_Parameters: -XGpb_Build=Debug -d
-P/home/sr/Seafire/Sowebio/informatique/github/gpb/gpb.gpr
-Xgpb_Build=Debug -s
Gprbuild_Project: /home/sr/Seafire/Sowebio/informatique/github/gpb/gpb.gpr
Gprbuild_Gpb_Scp: root@domain.tld:/usr/local/bin
Gprbuild_Gpb_Beep: bell
```

```
Compile
[Ada]      gpb.adb
[Ada]      v20.adb
[Ada]      v20-fls.adb
[Ada]      v20-log.adb
[Ada]      v20-prg.adb
[Ada]      v20-sys.adb
[Ada]      v20-tio.adb
[Ada]      v20-vst.adb
[Ada]      gnatcoll.ads
[Ada]      gnatcoll-memory.adb
Bind
[gprbind]  gpb.bexch
[Ada]      gpb.ali
Link
```

[link] gpb.adb

```
Gprbuild_Binary_Orig:
/home/sr/Seafire/Sowebio/informatique/github/gpb/bin/gprbuild
Gprbuild_Binary_Dest: root@domain.tld:/usr/local/bin/gprbuild
```

```
SCP copy (2638Kb) in progress...
SCP copy (15s @ 175Kbps) successful.
```

```
[2021-12-14 14:52:25] process terminated successfully, elapsed time: 16.32s
Output saved in
/home/sr/Seafire/Sowebio/informatique/github/gpb/obj/debug/messages.txt
```

Switching between local and remote copies can be done without leaving GnatStudio, from the "Build All" window, by choosing in the "drop down" list at the bottom of the window, the desired string, then click on [Execute].

It will be the choice, in the "drop down" list, between [for example] :

```
%builder -XGpb_Build=Debug -d %eL -P%PP %config %autoconf %X -s
```

```
%builder -XGpb_Build=Debug -XGpb_Scp=root@i51c1.xxx.org:/usr/local/bin
-d %eL -P%PP %config %autoconf %X -s
```

4.5 Open the project

Click on gpb.gpr to open the project in GNATStudio.

4.6 Build Gpb

Proceed as for AIDE building.

4.7 Installation

Example :

- Rename ~/gnat-20xx/bin/gprbuild to gprbuild_exe

- Copy ~/gpb/bin/gprbuild to ~/gnat-20xx/bin/gprbuild

4.8 Host authenticity with SCP

Using instance.domain.tld, if you get this kind of message :

```
The authenticity of host 'instance.domain.tld (***.***.***.***)' can't be established.
```

Use the command ssh-copy-id :

```
user@system: ssh-copy-id i51c1.genesix.org
```

5 Documentation

The documentation in various formats [PDF, PS, HTML or Texinfo] is available here : <https://gcc.gnu.org/onlinedocs>

Choose the corresponding GCC version. In our context, it is the last one [end of 2023]: version 13.2

These manuals below have a total of around 2000 pages. GNAT is well documented:

- GCC 13.2 Manual.pdf
- GCC 13.2 GNAT User's Guide.pdf
- GCC 13.2 GNAT Reference Manual.pdf
- GNAT Coding Style Manual.pdf [at the end of the web page]



GNAT Studio full workflow

There are 10 types of people in the world: those who understand binary and those who don't.

Anonymous



1 Project example Hello from Alire repository

1.1 Get Hello

```
user@system : cd $HOME/opt/alire

user@system : alr get hello

Clonage dans '/home/sr/.config/alire/indexes/community/repo'...
remote: Enumerating objects: 7784, done.
remote: Counting objects: 100% (94/94), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 7784 (delta 29), reused 72 (delta 16), pack-reused 7690
Réception d'objets: 100% (7784/7784), 1.33 Mio | 9.30 Mio/s, fait.
Résolution des deltas: 100% (4228/4228), fait.
① Deploying hello=1.0.2...
Clonage dans '/home/sr/opt/alire-test/alr-olkw.tmp'...
remote: Enumerating objects: 34, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 34 (delta 0), reused 10 (delta 0), pack-reused 24
Réception d'objets: 100% (34/34), 5.44 Kio | 5.44 Mio/s, fait.
Résolution des deltas: 100% (5/5), fait.
① Deploying libhello=1.0.1...
Clonage dans '/home/sr/opt/alire-test/hello_1.0.2_5715870b/alire/cache/dependencies/alr-lwuj.tmp'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 30 (delta 0), reused 12 (delta 0), pack-reused 17
Réception d'objets: 100% (30/30), 5.20 Kio | 5.20 Mio/s, fait.
Résolution des deltas: 100% (3/3), fait.

hello=1.0.2 successfully retrieved.
Dependencies were solved as follows:

+ libhello 1.0.1 (new)

user@system : ls -l

drwxrwxr-x 11 dv dv 4096 mai 29 11:51 gnat_native_12.2.1_11f3b811
drwxrwxr-x 6 dv dv 4096 mai 29 11:49 gprbuild_22.0.1_24dfc1b5
drwxrwxr-x 8 dv dv 4096 mai 29 12:03 hello_1.0.2_5715870b

user@system : cd hello_1.0.2_5715870b
```

1.2 GNAT Studio IDE

At the root of your alire project, open a terminal and execute:

`user@system : alr edit`

1.2.1 Clean all

GNAT Studio: Build > Clean > Clean all

1.2.2 Build all

GNAT Studio: Build > Project > Build all [F9]

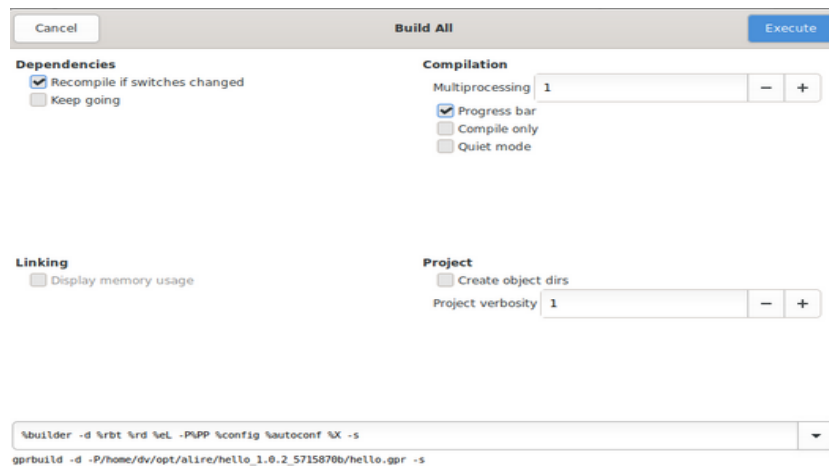
In the build window, check the build string:

```
%builder -d %rbt %rd %eL -P%PP %config %autoconf %X
```

Which is automatically translated by GNAT Studio as:

```
gprbuild -d -P/home/dv/opt/alire/hello_1.0.2_5715870b/hello.gprlash_led.gpr
```

Press [Execute] to build:



Build log:

```
gprbuild -d -P/home/dv/opt/alire/hello_1.0.2_5715870b/hello.gpr -XLIBHELLO_LIBRARY_TYPE=static -XLIBRARY_TYPE=static -XADAFLAGS=
Compile
  [Ada]      hello.adb
  [Ada]      libhello_config.ads
  [Ada]      libhello.adb
Build Libraries
  [gprlib]   Libhello.lexch
  [archive]  libLibhello.a
  [index]    libLibhello.a
Bind
  [gprbind]  hello.bexch
  [Ada]      hello.ali
Link
  [link]     hello.adb
[2023-05-30 13:56:27] process terminated successfully, elapsed time: 01.10s
```

2 Create native Alire project

2.1 Create Alire project

Select a root working directory:

```
user@system : cd $HOME/Sowebio/devgpl/stagiaires/arthur
```

Initialize Alire project in prog01 sub-directory:

```
user@system : alr init --bin prog01
```

Alire needs some user information to initialize the crate author and maintainer, for eventual submission to the Alire community index. This information will be interactively requested now.

You can edit this information at any time with 'alr config'

Please enter your GitHub login: (default: 'github-username')

>

Using default: 'github-username'

✓ prog01 initialized successfully.

2.2 GNAT Studio IDE

At the root of this alire project, open a terminal and execute:

```
user@system : cd $HOME/Sowebio/devgpl/stagiaires/arthur/prog01
```

```
user@system : alr edit
```

2.2.1 Build all

GNAT Studio: Build > Project > Build all [F9]

In the build window, check the build string:

```
%builder -d %rbt %rd %eL -P%PP %config %autoconf %X -s
```

Which is automatically translated by GNAT Studio as:

```
gprbuild -d -P/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/prog01.gpr -s
```

Press [Execute] to build:

Cancel

Build All

Execute

Dependencies
☒ Recompile if switches changed
☐ Keep going

Compilation

Multiprocessing 1 - +

☒ Progress bar
☐ Compile only
☐ Quiet mode

Linking
☐ Display memory usage

Project
☐ Create object dirs

Project verbosity 1 - +

gprbuild -d %rbt %rd %el -P%PP %config %autoconf %X -s

gprbuild -d -P/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/prog01.gpr -s

Build log:

```
gprbuild -d -P/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/prog01.gpr -XADAFLAGS= -s
Setup
[mkdir]          object directory for project Prog01
Compile
[Ada]            prog01.adb
prog01.adb:20:16: (style) bad casing of "Text_IO" declared at a-textio.ads:58
prog01.adb:23:01: (style) trailing spaces not permitted
prog01.adb:25:01: (style) trailing spaces not permitted
prog01.adb:29:01: (style) trailing spaces not permitted
prog01.adb:31:04: (style) space required
Bind
[gprbind]        prog01.bexch
[Ada]            prog01.ali
Link
[link]           prog01.adb
[2023-06-05 16:54:22] process terminated successfully, elapsed time: 01.50s
```

Builder results:

```
/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/src/prog01.adb
20:16 (style) bad casing of "Text_IO" declared at a-textio.ads:58
23:1 (style) trailing spaces not permitted
25:1 (style) trailing spaces not permitted
29:1 (style) trailing spaces not permitted
31:4 (style) space required (2 spaces mandatory after --)
```

2.3 Run

GNAT Studio: Build > Run > Run Main Prog01 [Maj]+[F2]

In the run window, check the box:

[x] Run in executables directory

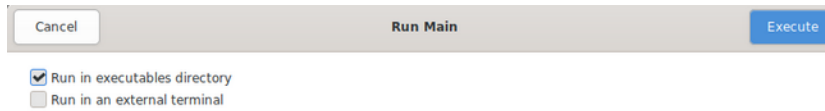
In the Run Main window, check the run string:

[exec_dir] %E

Which is automatically translated by GNAT Studio as:

```
/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/bin/prog01
```

Press [Execute] to run Prog01:



A dialog box with a title bar. It contains three buttons: 'Cancel', 'Run Main', and 'Execute'. Below the buttons, there are two checkboxes: 'Run in executables directory' (checked) and 'Run in an external terminal' (unchecked).



A terminal window with a dropdown menu showing '[exec dir] xE'. Below the dropdown, the command path is displayed: '/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/bin/prog01'.

Run Main results:

```
/home/dv/Sowebio/devgpl/stagiaires/arthur/prog01/bin/prog01
Hello World!
[2023-06-05 16:53:31] process terminated successfully, elapsed time: 00.31s
```

3 Convert a project to Alire

Kalle is a small indexed btree database from Wasiliy W. Molostoff with 35 files and 5 test programs.

Kalle is available at: <https://github.com/sowebio/aide-repository>

In directory: examples/kalle

3.1 Alire Kalle repository conversion

At the root of your alire repository, open a terminal and execute:

```
user@system : cd $HOME/opt/alire
user@system : git clone https://github.com/sowebio/kalle
```

Then, at the root of this new repository, execute:

```
user@system : alr init --bin --in-place kalle
```

Alire needs some user information to initialize the crate author and maintainer, for eventual submission to the Alire community index. This information will be interactively requested now.

You can edit this information at any time with 'alr config'

```
Please enter your GitHub login: (default: 'github-username')
>
Using default: 'github-username'
Please enter your full name: (default: 'Your Name')
> Stéphane Rivière
Please enter your email address: (default: 'example@example.com')
> sriviere@soweb.io
✓ kalle initialized successfully.
```

Thus, new files were generated:

```
user@system : tree -L 1
```

```
kalle/
├── alire
├── alire.toml
├── bin
├── config
├── kalle.gpr
├── kalle.txt
├── obj
├── share
└── src
```

3.2 Alire Kalle repository customization

Let's customize it:

```
user@system : rm ./alire/kalle/src/kalle.adb

user@system : sed -i 's/("src\/",/("src\/**",/g' ./alire/kalle/kalle.gpr
user@system : sed -i 's/("kalle.adb")/("test_ah.adb", "test_btav.adb", "test_btpa.adb",
"test_dbase.adb", "test_fs.adb")/g' ./alire/kalle/kalle.gpr
user@system : sed -i 's/Ada_Compiler_Switches/Ada_Compiler_Switches \& "-gnat95"/g' ./alire/kalle/
kalle.gpr
```

3.3 Kalle build

```
user@system : cd $HOME/opt/alire/kalle
user@system : alr edit
```



Cancel

Build All

Execute

Dependencies

☒ Recompile if switches changed
 ☐ Keep going

ue as much as possible after a compilation error

Compilation

Multiprocessing 1

☒ Progress bar
 ☐ Compile only
 ☐ Quiet mode

Linking

☐ Display memory usage

Project

☐ Create object dirs

Project verbosity 1

%builder -d %rbt %rd %eL -P%PP %config %autoconf %X -s

gprbuild -d -P/home/dv/opt/alire/kalle/kalle.gpr -s

3.4 Kalle tests programs

```

user@system : ~/opt/alire/kalle/bin$ ./test_ah

--- try-suffix: true: ordinary
--- try-insert: true: with expand for 1 elem
--- try-insert: true: with amend for 1 elem
--- try-insert: true: with delete for 1 elem
--- try-insert: true: with delete
--- try-insert: true: with expand
--- try-expand: true: ordinary
--- try-delete: true: ordinary
--- try-locate: true: full subpattern matching

user@system : ~/opt/alire/kalle/bin$ ./test_btav

--- try-insert: true: inserting values in empty tree
--- try-get_ge: true: try to get value over the upper bound
--- try-get_ge: true: find values by less argument
--- try-get_ge: true: find values by equal argument
--- try-get_le: true: try to get value over the lower bound
--- try-get_le: true: find values by equal argument
--- try-get_le: true: find values by greater argument
--- try-get_lt: true: try to get value over the lower bound
--- try-get_lt: true: find values by greater argument
--- try-get_lt: true: find values by equal argument
--- try-get_gt: true: try to get value over the upper bound
--- try-get_gt: true: find values by less argument
--- try-get_gt: true: find values by equal argument
--- try-delete: true: 1st part
--- try-delete: true: 2nd part

user@system : ~/opt/alire/kalle/bin$ ./test_btpa

--- try-insert: true: inserting values in empty tree
--- try-get_ge: true: try to get value over the upper bound
--- try-get_ge: true: find values by less argument
--- try-get_ge: true: find values by equal argument
--- try-get_le: true: try to get value over the lower bound
--- try-get_le: true: find values by equal argument
--- try-get_le: true: find values by greater argument
--- try-get_lt: true: try to get value over the lower bound
--- try-get_lt: true: find values by greater argument
--- try-get_lt: true: find values by equal argument
--- try-get_gt: true: try to get value over the upper bound
--- try-get_gt: true: find values by less argument
--- try-get_gt: true: find values by equal argument
--- try-delete: true: 1st part
--- try-delete: true: 2nd part

user@system : ~/opt/alire/kalle/bin$ ./test_dbase

```

```
user@system : ~/opt/alire/kalle/bin$ ./test_fs
```

```
[ 40: 50: 60]
[ 40: 50: 0 60]
]%
```

4 Convert a library to Alire

V20 is an example of library which has a dependency with another library called GNATColl.

V20 is available at : <https://github.com/sowebio/v20>.

4.1 Alire repository conversion

To get the repository, open a terminal and execute:

```
user@system : cd $HOME/opt/alire
user@system : git clone https://github.com/sowebio/v20
```

Then, at the root of this new repository, execute:

```
user@system : alr init --lib --in-place --no-skel v20
```

Alire needs some user information to initialize the crate author and maintainer, for eventual submission to the Alire community index. This information will be interactively requested now.

You can edit this information at any time with 'alr config'

Please enter your GitHub login: (default: 'github-username')

>

Using default: 'github-username'

Please enter your full name: (default: 'Your Name')

> Stéphane Rivière

Please enter your email address: (default: 'example@example.com')

> sriviere@soweb.io

✓ v20 initialized successfully.

The standard Alire tree is generated alongside previous files:

```
user@system : tree -L 1
v20-alire
├── alire
├── alire.toml
├── bak
├── bin
├── config
├── doc-generated
├── nohup.out
├── obj
├── README.md
├── src
├── src-out
├── src-tests
├── v20.aru
├── v20.copyrights
├── v20.dbg
├── v20.gpr
├── v20.html
├── v20.txt
└── v20.udb
```

4.2 Setup dependencies with Alire

V20 requires the GNATColl library, which can be linked using the following command:

```
user@system : alr with gnatcoll
```

4.3 Edit and build Alire library

```
user@system : alr edit
```

Cancel

Build All

Execute

Dependencies
☒ Recompile if switches changed
☐ Keep going

Compilation
Multiprocessing - +
☒ Progress bar
☐ Compile only
☐ Quiet mode

Linking
☐ Display memory usage

Project
☐ Create object dirs
Project verbosity - +

%builder -d %rbt %rd %eL -P%PP %config %autoconf %X -s

▼

4.4 Test library

```
user@system : cd bin && ./test
```

```
v20 library test program - test v0.6  
Copyright (C) Sowebio SARL 2020-2021, according to GPLv3.  
  
20230609-114550 - INIT      - MSG - Ada Cur: [ 1388 ] Max: [ 1696 ]  
20230609-114550 - INIT      - MSG - All Cur: [ 2969600 ] Max: [ 2969600 ]  
...
```



<https://this-page-intentionally-left-blank.org>



Learning Ada

Doubling the number of programmers on a late project does not make anything else than double the delay.

Second Brook's Law



1 Introduction

Ada is not just programming, Ada is software engineering.

Before study Object Oriented Programming, study first Structured Programming.

2 Requirements

GNAT Toolchain.

3 Historical books

Structured Programming – Dahl, Dijkstra, Hoare [1972]

Principle of Program Design – Jackson [1975]

A Structured programming Approach to Data – Coleman [1978]

4 Ada books

Ada avec le Sourire – Bergé, Donzelle, Olive, Rouillard [1989]

Méthodes de Génie Logiciel avec Ada 95 – Rosen [1995] - https://fr.wikibooks.org/wiki/M%C3%A9thodes_de_g%C3%A9nie_logiciel_avec_Ada

Ada Essentials: Overview, Examples and Glossary – Crawford [2000]

Ada distilled – Riehle [2003] - <https://www.sigada.org/wg/eduwg/pages/Ada-Distilled-07-27-2003-Color-Version.pdf>

Adacore books: multiple authors - <https://www.adacore.com/books>

Ada Development Environment on Linux

5 Ada courses

Ensemble pédagogique IUT - Feneuille [2003] - <http://d.feneuille.free.fr/paquetage.htm>

6 Ada links

Le langage Ada: https://www.adalog.fr/fr/faq_ada.html



Coding examples

Variables won't; Constants aren't.
Osborn Law



1 HAC Ada interpreter

If you're not an experienced programmer, we invite you to use HAC, an outstanding Ada subset interpreter.

HAC is available at: <https://github.com/zertovitch/hac>

HAC documentation [source] is available at: <https://github.com/sowebio/hac-doc>

2 GNATStudio Examples

2.1 Drink dispenser

<<<TODO>>>

2.2 Sorting algorithm

<<<TODO>>>

3 GNATStudio Examples [AVR 8 bits microcontroller]

Ada Development on 8 bits AVR Microcontroller [ADAM] is based on the latest GNAT, Alire and GNAT Studio releases and allows real-time AVR debugging in GNAT Studio.

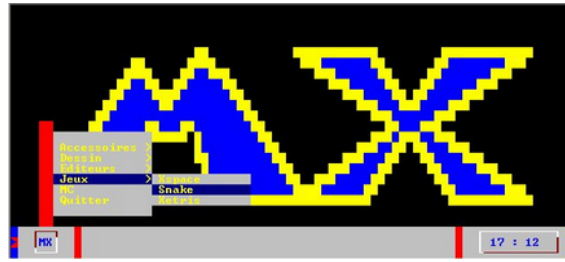
ADAM is available at: <https://github.com/sowebio/adam-doc>

4 Programs from the MX Team

Theses programs are available at: <https://github.com/sowebio/aide-repository>

In directory: examples/aide/projects/mx-team

4.1 Mx



4.1.1 Overview

Mx was coded by Xavier, 13 years old in 2004, when he discovered programming and Ada five months before. Mx is an application launcher.

The Start" button, named here "Mx" is in relief. The menus are nested. Mx uses the '.vsl' resource files created by Visual. Visual also use Mx as a main program.

4.1.2 Build

The sources of Mx are available in:

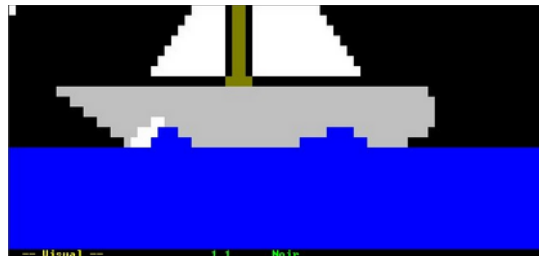
<<<TODO>>>

4.1.3 Usage

- General commands

- | | | |
|-------------------|------|------------|
| - [Esc] | Exit | |
| - [Enter ↵] | | Validation |
| - [←] [↑] [→] [↓] | Move | |

4.2 Visual



4.2.1 Overview

Visual was coded by Martin, 13 years old in 2004, when he discovered programming and Ada five months before.

Visual is a text-based screen editor. The created images can be saved in screen image files with the extension '.vsl'. These files can be used directly as external resources by third party applications.

4.2.2 Build

The sources of Visual are available in:

<<<TODO>>>

4.2.3 Usage

- General commands

- | | | |
|-------------------------|------|--------------|
| - [Esc] or [Alt] + [F4] | Exit | |
| - [Ctrl] + S | | Save to file |
| - [Ctrl] + O | | Open a file |
| - [Ctrl] + N | | New file |

- Selection of the "brush" colors

- | | |
|-----------------|---------------|
| - [F1] | Black |
| - [F2] | Blue |
| - [F3] | Green |
| - [F4] | Cyan |
| - [F5] | Red |
| - [F6] | Magenta |
| - [F7] | Brown |
| - [F8] | Grey |
| - [F9] | Yellow |
| - [F10] | White |
| - [Ctrl] + [F2] | Light blue |
| - [Ctrl] + [F3] | Light green |
| - [Ctrl] + [F4] | Light cyan |
| - [Ctrl] + [F5] | Light red |
| - [Ctrl] + [F6] | Light magenta |

4.3 Updates from original 2004 release

4.3.1 Overview

MX team programs were developed in 2004 and tested under Windows 2K only, using some functions from the v04 library, a console multi-platform library for Windows and ANSI console.

v04 library has been resurrected and then simplified to use ANSI console only. Some Windows special features were hardcoded in MX team programs to get graphic effects. They have been slightly modified to handle this new environment.

<<<TODO>>>

<https://this-page-intentionally-left-blank.org>



Programming basics

Weinberg's Second Law : If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
Gerald Weinberg



Ada is very well suited for educational purposes. If you are not an experienced programmer mastering a procedural method programming as a tool, you may find this chapter useful. Your creative spirit's the limit.

If you are not an experienced programmer mastering a method programming as a tool, you may find this chapter useful. Your creative spirit's the limit.

This chapter deals with top-down analysis and modular programming method. Understanding and assimilating the following will already make you a very good developer.

This matter is not an end but a foundation to go further. Like understanding the differences between object programming by classification or by composition. And why, for many projects, object programming should be avoided and for others, it should really be adopted.

So, no object methods will be discussed. It's beyond the scope of this manual. Most developers using object-oriented languages have not learned any methods, using wrong tools with no thinking. We know the result.

To be good at object-oriented development, you must already understand the basics of analysis and modular and structured programming.

One step after the other :]

1 Tools

To create a program, you must:

- Master an analytical method;
- Know Boolean algebra;
- Use a programming language;
- Have a good general culture and know-how.

Ada Development Environment on Linux

Of these four elements, the first one is the most difficult to acquire, but I hope that the following lines will help you in this field.

I could have added: paper, a pencil and an eraser, because these three objects are always the basis of a good program and you should not rush to code.

You will notice that the knowledge of a language comes after the theory. This is normal. As analysis precedes writing, mastering design precedes mastering a language. Finally...

The joy of programming must remain the driving force of your motivation.

2 Analysis

2.1 Methods overview

The main classes of methods are:

- Modular and structured programming ;
- Object method by composition
- Object method by classification.

The modular and structured programming method is still used in many fields as the main programming method.

It is also used in object methods, at least in the following contexts:

- In the main startup and finalization module;
- In the functions [methods] of the objects.

Object method can be divided into object methods by composition or by classification.

The object method by classification [hierarchical] is the most known object method *and yet the least relevant*, except for developing a graphical interface or any other project clearly requiring the inheritance tool.

Object method programming is beyond the scope of this manual.

2.2 Top-Down example

The top-down analysis approach is one among many. It is intuitive and efficient. One can fly rockets with it but it is good that you know that other ways exist.

Everyone programs, the car mechanic, the postal worker and the cook. Didn't you know that? So let's start by cooking an egg!

Mastering an analysis method allows to *analyze a problem*, even a very complex one, and break it down by *successive refinements*, into a sum of problems, one by one so obvious to solve, that one stops the analysis by declaring it is finished!

So we're going to cook an egg, a hard-boiled egg to be precise. But could you detail such a seemingly simple process without hesitation? Let's see it together.

2.2.1 Problem's decomposition

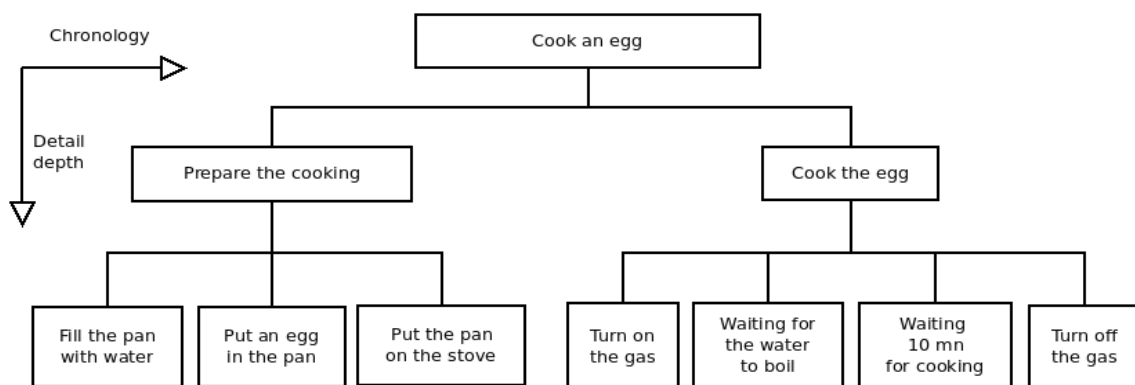
We could, for example, start by breaking down, by *refining*, the action of cooking an egg into two main steps two main steps: *preparation* and *cooking*.

Then we could take these two main steps and refine them again:

- The preparation is to fill the pan with water, put an egg in it and put the pan on the stove;
- Cooking is turning on the gas, waiting for the water to boil, wait 10 minutes for cooking¹⁵, then turn off the gas.

This approach is known as *decomposition by successive refinements*.

Once this decomposition is completed, it is essential to represent it visually, thanks to the *PSD*, the *Program Structure Diagram*, sometimes called the *JSP Structure Diagram*, after its inventor¹⁶:



This PSD works in two dimensions:

- In the vertical plane, we go down from the most complex to the simplest;
- In the horizontal plane, the direction of the reading represents naturally, chronologically, the tasks to be performed.

The PSD has a dual purpose:

- In the first instance, it allows you to gain an *overview of the problem at hand* and ensure that your analysis is *consistent and complete* ;

¹⁵ It's a lot, but not a problem, unless you like them soft. The shell will come off more easily.

¹⁶ It is difficult to determine the origin of these concepts. Many researchers worked on them at the same time. One of Jackson's merits was to promote the notion of initial read-current read in loop processing. https://en.wikipedia.org/wiki/Jackson_structured_programming

- Secondly, since each box represents an action that is so simple to solve that it does not require further analysis, the PSD allows you to go directly to the second phase: *the pseudo-code!*

In creating this PSD, we have *modularized* our problem. We have *decomposed* our problem into a series of *elementary modules*. When writing the *pseudo-code*, we will describe the functioning of each *module* using *structures*. These *structures* form the basic building blocks of *structured* programming, without goto or spaghetti code.

2.2.2 Pseudo-code

The pseudo-code is the computer translation, as structures, of the already written PSD.

The PSD and the pseudo-code are linked. They must be consistent with each other.

It is often while checking this consistency, at the time of writing the pseudo-code, that one realizes that the level of detail of the PSD is incorrect. If the level of detail is too high, the pseudo-code contains useless modules that do not contain any processing that deserves to be modularized. On the other hand, if the level of detail is not high enough, the pseudo-code contains modules that are far too big.

Before going into the details of the general writing of a pseudo-code, let's see a small example, with our hard-boiled egg, just to get a taste of it.

```
begin *** cook an egg ***  
  
do *** prepare the cooking ***  
do *** cook the egg ***  
  
end *** cook an egg ***
```

In this first pseudo-code, representing the main module of the "cook an egg" program, the analogy between PSD and pseudo-code is clear.

The term *do* before *prepare the cooking* represents the call to the module *prepare the cooking*. Each module starts with *start *** module name **** and ends with *end *** module name ****.

Let's move on to writing prepare the cooking module:

```
begin *** prepare the cooking ****  
  
do while "pan is not filled"  
fill with water  
end do while  
  
do *** put the pan on the stove ***  
  
do *** put an egg in the pan ***  
  
end *** prepare the cooking ***
```

This is when a problem arises. The module putting the pan on the stove is a really very simple action. A so simple one that it does not, in fact, deserve to be isolated in a module. Leaving the analysis as it is, without changing anything, would result in making the program more complex than it deserves to be.

So we will simplify the pseudo-code:

```
begin *** prepare the cooking ****
  do while "pan is not filled"
    fill with water
  end do while

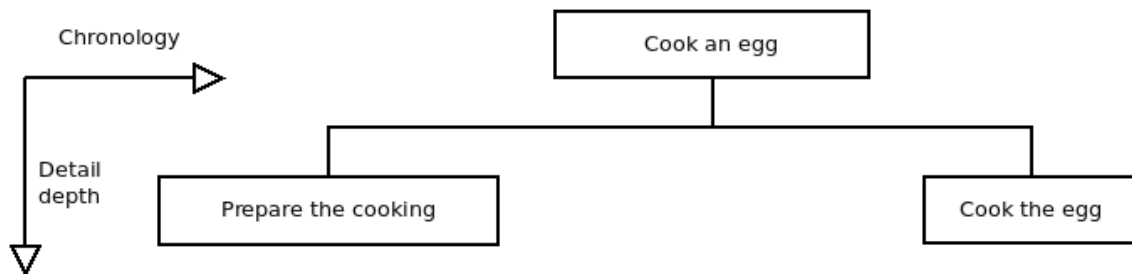
  put the pan on the stove

  put an egg in the pan

end *** prepare the cooking ***
```

So it appeared that the level of detail in the PSD was too high. The actions of the last rank: *pan on the fire*, *fill with water*, etc. did not deserve, by themselves, a separate module.

They should be grouped together in the modules of higher rank: *prepare the cooking* and *cook the egg*.



The analysis of *the cooking of the egg* ends with the pseudo-code of the last module:

```
begin *** cook the egg ***
  turn on the gas

  do while "water does not boil"
    wait
  end do while

  do while "not 10 minutes elapsed"
    wait
  end do while

  turn off the gas

end *** cook the egg ***
```

After this example, we now take a closer look at this analysis method.

3 Modular and structured programming method

3.1 Introduction

This modular and structured programming approach is generic to dozens of methods invented in the 1980s to make software execution more reliable and improve maintenance.

These methods differed essentially in the symbols, vocabulary and aesthetics of the diagrams. They are still relevant today as the indispensable basis of the methods used by a good developer.

The method illustrated here is GMSP: General, Modular and Structured Programming¹⁷. It comes from the teaching provided by the french Control Data Institute, located in Paris, which has now disappeared, with the help of PLATO¹⁸, a Computer Aided Learning system. Graphical extensions to these methods exist, for example SADT or its real-time extension SART.

The author does not really appreciate graphical representations [which make nice drawings for IT managers] in analysis methods. Flowcharts, flow diagrams, SADT or UML graphs generally bring more confusion than information.

However, some graphical representations, such as the PSD or the HOOD method diagrams, are good tools. They are the first steps of the written specifications, which can be found, strictly speaking, in the specifications of an Ada package.

3.2 Program Structure Diagram

Writing a PSD - *Program Structure Diagram* - means identifying, decomposing and prioritizing functions in a coherent whole, in order to allow the writing of the program pseudo-code.

3.2.1 Process detailed

The process of creating the PSD is an iterative one, which loops around itself, to identify all the tasks to be carried out, until the possibilities of refinement are exhausted, i.e. until the problem to be solved can no longer be detailed.

This approach is called a *top-down approach*, in order to show that we start from the global problem, at *the top of the diagram*, and work our way down to the smallest detail, *towards the bottom of the diagram*. Each time we add a level of detail, we create a new line.

For each detail level, the identified tasks are written in the reading direction, in order of execution. They are placed in *boxes*. For clarity of the PSD, all boxes of a lower rank are connected by lines to the box of the higher rank.

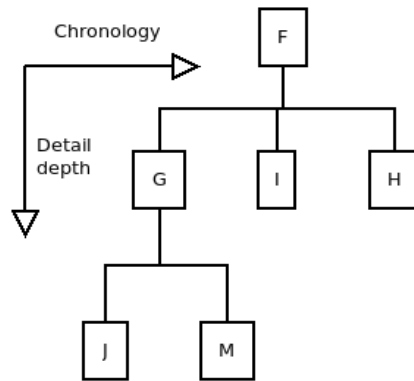
PSDs are always written and read:

- Top to bottom, for level of detail;
- From left to right, for chronological steps.

Example:

¹⁷ PGMS in french, as “Programmation Générale, Modulaire et Structurée”

¹⁸ Programmed Logic for Automatic Teaching Operations - [https://en.wikipedia.org/wiki/PLATO_\(computer_system\)](https://en.wikipedia.org/wiki/PLATO_(computer_system))



In no case does a PSD show the tests and other low-level actions that are the responsibility of programming.

A PSD is both the overview and the backbone of the analysis.

Writing the PSD is the most difficult part of the analysis.

3.3 Pseudo-code

The pseudo-code writing is done from the PSD. Each *box* of the PSD will correspond to a module in the *pseudo-code*.

⇒ We repeat: one PSD box to one module in the pseudo-code.

The writing of a pseudo-code is done from elementary bricks, which we will examine now.

3.3.1 Main module

A program *starts* and *ends* at the master module.

Here is the pseudo-code, also called PC, of the previous PSD, describing the master module of program F:

```

begin *** F ***
  do *** G ***
    do while P (while P is true)
      do *** I ***
    end do while
  do *** H ***
end *** F ***

```

The beginning of a module is represented by `begin *** module name ***` and the end of a module is represented by `end *** module name ***`.

The name of the main module is the name of the program.

3.3.2 Other modules

Other modules are written the same way. Here is the pseudo-code of module G of the previous PSD, describing the program G:

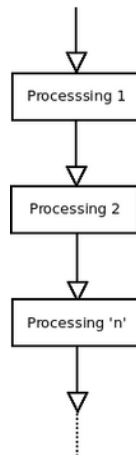
```

begin *** G ***
do *** J ***
if Q (if Q is true)
do *** M ***
end if
end *** G ***

```

3.3.3 Sequence

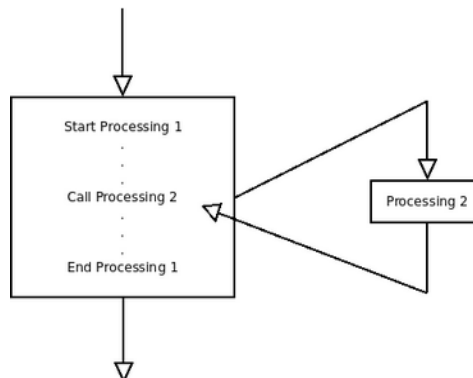
Sequence is the simplest form of pseudo-code. It just represents the sequence of several processes, which are executed one after the other:



3.3.4 Module call

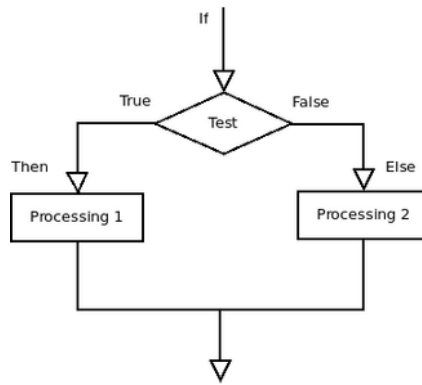
Module call is represented by `do *** module name ***`. The processing of the calling module stops at the line of the call and the called module executes.

At the end of the called module, the latter returns to the calling module and the execution of the latter resumes at the line following the call which has just been executed:



3.3.5 If... else... end if

The alternative is the simplest test of a pseudo-code. Depending on the truth of the test condition, the program flow is directed to one processing or another:



The alternative is represented in pseudo-code as follows:

```

if test condition (is true)
  Processing 1
else
  Processing 2
end if
  
```

❑ If... elsif... else... endif

This structure is an extension of the alternative:

```

if test condition 1
  Processing 1
elsif test condition 2
  Processing 2
elsif test condition 3
  Processing 3
else
  Default processing
end if
  
```

The default processing is executed when no test condition has been checked.

This structure is equivalent to a nesting of alternatives. But these nestings are much less readable, as shown in the example below:

```

if test condition 1
  Processing 1
else
  if test condition 2
    Processing 2
  else
    if test condition 3
      Processing 3
    else
      Default processing
    end if
  end if
end if
  
```

3.3.6 Case... when... else... end case

The selection is a different form of the alternative because the test is no longer Boolean [true or false] but depends on the content of the tested value. A pseudo-code is more meaningful:

```

selection value to test

when value 1
  Processing 1

when value 2
  Processing 2

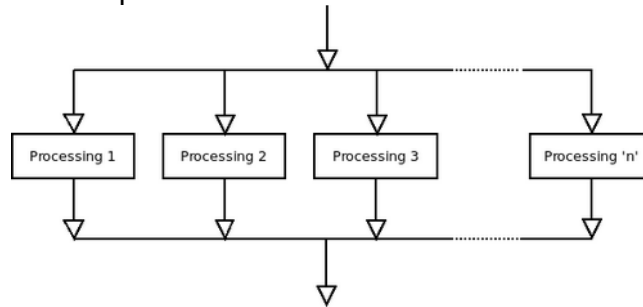
when value 3
  Processing 3

when others
  Default processing

end selection

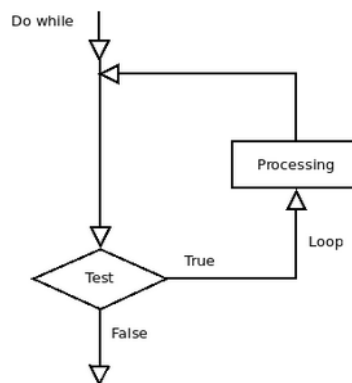
```

This structure can be represented as follows:



3.3.7 Do while... end do

This loop structure is useful when you want the program flow *to avoid processing in the loop if the condition is false at the first pass in the loop*:



The pseudo code of such a structure is as follows:

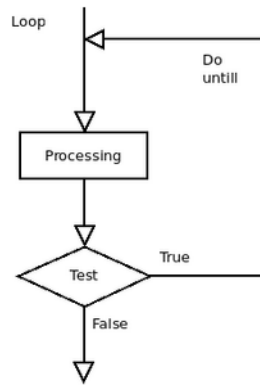
```

do while test (is true)
  process
end do while

```

3.3.8 Loop... until

This loop structure differs from the previous one because the processing in the loop is done once before the loop condition is tested. Thus, one will always pass at least once in this type of loop:



Here is the notation of the loop... until in pseudo-code:

```

loop
  process
until condition test (is true)
  
```

It is clear that the test is performed after a first pass in the loop.

3.4 Functions

➤ One point of entry, one point of exit. No anticipated exit. Never. We repeat: never :)

All parameters will be named and, if the language - such as Ada - allows it, the parameter names will be used in the function calls.

4 Boole algebra

Here is a practical summary about Boolean algebra, which should be known by all developers.

4.1 Identities, properties and De Morgan's laws

Two conventions are used:

- \equiv for equivalence. $A \equiv B$ means that A and B are two equivalent conditions and that they are interchangeable;

- NOT A for the negation of A. If A is true, NOT A is false.

4.1.1 Identities

$A \text{ OR } 0 \equiv A$	$\text{NOT} (\text{NOT } A) \equiv 1$
$A \text{ OR } 1 \equiv 1$	$A \text{ OR } A \equiv A$
$A \text{ OR } (\text{NOT } A) \equiv 1$	$A \text{ AND } A \equiv A$
$A \text{ AND } (\text{NOT } A) \equiv 0$	

4.1.2 Properties

$A \text{ AND } B \equiv B \text{ AND } A$
$A \text{ OR } B \equiv B \text{ OR } A$
$A \text{ AND } (B \text{ AND } C) \equiv (A \text{ AND } B) \text{ AND } C$
$A \text{ OR } (B \text{ OR } C) \equiv (A \text{ OR } B) \text{ OR } C$
$A \text{ AND } (B \text{ OR } C) \equiv (A \text{ AND } B) \text{ ET } (A \text{ AND } C)$
$A \text{ AND } (B \text{ OR } C) \equiv (A \text{ AND } B) \text{ ET } (A \text{ AND } C)$

4.1.3 De Morgan's law

$\text{NOT } (A \text{ OR } B) \equiv (\text{NOT } A) \text{ AND } (\text{NOT } B)$
 $\text{NOT } (A \text{ AND } B) \equiv (\text{NOT } A) \text{ OR } (\text{NOT } B)$

4.2 Practical advises

In your current language manual, you will certainly find the description of priorities in the evaluation of logical expressions.

The following is an example of evaluation priorities:

1. Expressions located in the innermost brackets;
2. Negation;
3. AND and OR [In the Ada language, these two operators are on an equal footing, which is not the general rule in other languages where AND usually has a higher priority than OR];
4. With equal priority, evaluate expressions from left to right.

✦ One might be tempted to take these priorities into account to write the shortest possible test condition, but *this should be avoided at all costs* for reasons of clarity.

Here are three basic rules *to follow in all circumstances*:

1. Never hesitate to *use parentheses* to increase readability and reliability.
2. To work on or reverse a complex condition, you must *first restore the implicit parentheses*.
3. A simplification of a complex condition is *done by applying the De Morgan's laws*.

5 Basics algorithms

5.1 Initial reading & current reading in loops

<<<TODO>>>

FAQ

With the Wildebeest and the Penguin, there's no Bull.
Number Six



1 Issues & solutions

1.1 Error when trying to reading documentation: No HTML browser specified

Q: I see theses errors in message console :

```
Launching xdg-open to view file:///home/sr/opt/gnat-2020/share/doc/gnatstudio/html/tutorial/index.html
[2021-03-14 21:45:08] No HTML browser specified
[2021-03-14 21:45:17] No HTML browser specified
[2021-03-14 21:45:37] No HTML browser specified
[2021-03-14 21:49:04] No source file selected
Launching /usr/bin/firefox to view file:///home/sr/opt/gnat-2020/share/doc/gnatstudio/html/tutorial/index.html
Launching /usr/bin/firefox to view file:///home/sr/opt/gnat-2020/share/doc/gnatstudio/html/users\_guide/index.html
```

A: Sets the real path of your browser of choice:

Edit > Preferences > External Commands > Browser > HTML Browser :
`/usr/bin/firefox %u`

1.2 No GNATStudio icon in dock

Check `~/.local/share/applications/gnatstudio.desktop` or `/usr/share/applications/gnatstudio.desktop` :

```
[Desktop Entry]
Name=GnatStudio
Icon=/opt/gnatstudio/share/gnatstudio/icons/hicolor/32x32/apps/gnatstudio_logo.png
Exec=/opt/gnatstudio/bin/gnatstudio
Terminal=false
Type=Application
MimeType=application/x-adagpr
Categories=Development;
StartupWMClass=gnatstudio_exe
```

If missing, create it.

1.3 Association lost between .gpr project files and GNATStudio

Check GNATStudio icon in dock [see above].

If a .gpr file has been opened with a program other than GNATStudio, the association between .gpr project files and GNATStudio may have been lost.

Right-click on a .gpr file, choose Properties and go to the “Open With” tab, select GNATStudio and then click the [Reset] button. The original association with GNATStudio is restored.

If GNATStudio choice does not appear, check file: ~/.local/share/mime/packages/x-adagpr.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mime-type xmlns="http://www.freedesktop.org/standards/shared-mime-info" type="application/x-adagpr">
  <!--Created automatically by update-mime-database. DO NOT EDIT!-->
  <comment>GNAT Gprbuild file</comment>
  <glob pattern="*.gpr"/>
</mime-type>
```

Check file: ~/.local/share/mime/packages/application-x-adagpr.xml

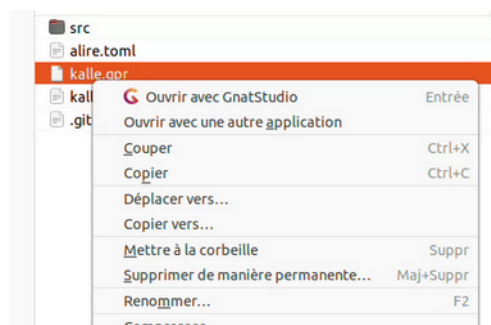
```
<?xml version="1.0" encoding="UTF-8"?>
<mime-info xmlns="http://www.freedesktop.org/standards/shared-mime-info">
  <mime-type type="application/x-adagpr">
    <comment>GNAT Gprbuild file</comment>
    <glob pattern="*.gpr"/>
  </mime-type>
</mime-info>
```

If one of these files are missing, create application-x-adagpr.xml and run:

[user@system](#) : update-mime-database

x-adagpr.xml will be created automatically.

Right click on a .gpr file, you must see “Open with GNATStudio”:



1.4 GNAT Runtime help tree is altered in GNATStudio

Q: Instead of having the package tree directly, you have to go through a whole path of intermediate menus before reaching the package menu.

A: You have initiated a run-time debugging session by uncomment the line

```
for Runtime ("Ada") use "/home/sr/opt/gnat-YYYY/lib/gcc/x86_64-pc-linux-gnu/X.Y.Z/rts-native-debug";
```

in the .gpr project file. This has the side effect to alter Menu > Help > GNAT Runtime tree package help files.

1.5 How file association is processed by the system

This is related to previous section above and list conditions summary for file type handling.

A GNATStudio launcher specifies the MIME¹⁹ type for the GNATStudio application:

```
~/.local/share/applications/gnatstudio.desktop

[Desktop Entry]
Name=GnatStudio
Icon=/opt/gnatstudio/share/gnatstudio/icons/hicolor/32x32/apps/gnatstudio_logo.png
Exec=/opt/gnatstudio/bin/gnatstudio
Terminal=false
Type=Application
MimeType=application/x-adagpr
Categories=Development;
StartupWMClass=gnatstudio_exe
```

The association file between extension .gpr and MIME type is done by this file:

```
~/.local/share/mime/packages/application-x-adagpr.xml

<?xml version="1.0" encoding="UTF-8"?>
<mime-info xmlns="http://www.freedesktop.org/standards/shared-mime-info">
  <mime-type type="application/x-adagpr">
    <comment>GNAT Gprbuild file</comment>
    <glob pattern="*.gpr"/>
  </mime-type>
</mime-info>
```

Finally, we update the MIME and Desktop databases:

```
MIME & Desktop DB updates
user@system: update-mime-database ~/.local/share/mime
user@system: update-desktop-database ~/.local/share/applications
```

By the way, a file is automatically generated:

```
~/.local/share/mime/application/x-adagpr.xml

<?xml version="1.0" encoding="utf-8"?>
<mime-type xmlns="http://www.freedesktop.org/standards/shared-mime-info" type="application/x-adagpr">
  <!--Created automatically by update-mime-database. DO NOT EDIT!-->
  <comment>GNAT Gprbuild file</comment>
  <glob pattern="*.gpr"/>
</mime-type>
```

Association test:

```
user@system: gio mime application/x-adagpr

Application par défaut pour « application/x-adagpr » : gnatstudio.desktop
Applications inscrites :
  gnatstudio.desktop
Applications recommandées :
  gnatstudio.desktop
```

¹⁹ MIME [IANA Types] stands for Multipurpose Internet Mail Extensions. For more information refers to: <https://datatracker.ietf.org/doc/html/rfc6838>.

1.6 Where are stored GNATStudio configuration files ?

Personal setting are located in:

```
~/ .gps (2019)
~/ .gnatstudio (2020 and later)
```

2 Ada

2.1 Check calls to external libraries

Use the LDD utility:

```
user@system: ldd ./test

linux-vdso.so.1 (0x00007ffcb9dd9000)
libz.so.1 => /home/sr/Seafire/Sowebio/informatique/dev/ada/lib/zlib-1211/contrib/ada/bin/../../../../
libz.so.1 (0x00007f3fcf111000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f3fcef0d000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3fceb1c000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3fcf32c000)
```

The line in bold is a link to a specific library.

If the program is statically linked:

```
user@system: ldd ./test

is not a dynamic executable
```

2.2 Library integration with .gpr

This is for your information as that's not the way we do things with Alire.

Check paths:

```
user@system: Cd ~/opt/alire/gnat_native_12.2.1_11f3b811/bin

user@system: gnat ls -v

GNATLS 12.2.0
Copyright (C) 1997-2022, Free Software Foundation, Inc.

Source Search Path:
<Current_Directory>
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/lib/gcc/x86_64-pc-linux-gnu/12.2.0/adainclude

Object Search Path:
<Current_Directory>
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/lib/gcc/x86_64-pc-linux-gnu/12.2.0/adalib

Project Search Path:
<Current_Directory>
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/x86_64-pc-linux-gnu/lib/gnat
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/x86_64-pc-linux-gnu/share/gpr
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/share/gpr
/home/dv/opt/alire/gnat_native_12.2.1_11f3b811/lib/gnat
```

2.3 Program calls analysis

Practical calls analysis. Useful to know which library is really called, and after which attempts. One will be surprised to see how many attempts a program can make before finding [or not] the wanted library:

```
user@system: sudo apt install strace ltrace
user@system: strace -o sortie.txt ./programme
user@system: strace -c ./programme
```

% time	seconds	usecs/call	calls	errors	syscall
38.64	0.004999	3	1451		write
33.58	0.004344	2	2718		read
10.69	0.001383	7	202	20	openat
6.89	0.000892	5	182		close
3.97	0.000513	1	363		fstat
2.93	0.000379	4	102		brk
2.27	0.000294	2	177		getcwd
0.39	0.000050	50	1		munmap
0.32	0.000042	8	5		rt_sigaction
0.19	0.000024	3	8		mprotect
0.07	0.000009	9	1		sigaltstack
0.06	0.000008	8	1		lseek
0.00	0.000000	0	16	14	stat
0.00	0.000000	0	10		mmap
0.00	0.000000	0	5	5	access
0.00	0.000000	0	1		execve
0.00	0.000000	0	1		readlink
0.00	0.000000	0	1		arch_prctl
100.00	0.012937		5245	39	total

In our case, we wanted to understand why the example program did not compile and therefore did not use the zlib library. The contributor to the demo program considered that the zlib library was installed by default at the system level. In the case of Ubuntu, via the package `zlib1g` [1=one].

2.4 Statically link an external library to an executable

To statically link zlib, you need to put the options below in the right order:

- First the search paths;
- Then the library or libraries.

Copy the static library `libz.a` to the current directory is allowed [or to `./obj` if the `Object_Dir` use 'obj' clause is used], but this is not a very clean way to proceed. It's better to use the path specification parameter `-L`.

The usage is, however, tricky:

- This parameter will not support any spaces or dots in the path;
- If both versions - shared and static - of the library exist in the same directory, the shared library `libz.so` will always be chosen over the static library `libz.a`;
- To force the choice of the static version, you must then specify by name the library to be statically linked with the `-l:libz.a` option instead of `-lz`.

Example:

```
-- gprbuild -d -P./zlib.gpr

project Zlib is

  for Languages use ("Ada");

  for Source_Dirs use ("src"); -- Avec parenthèses
  for Object_Dir use "obj"; -- Sans parenthèses

  for Main use ("test.adb", "mtest.adb", "read.adb", "buffer_demo");

  -- gnatmake
  --
  -- -gnat w cfilopru      Warnings management
  -- -gnat V cdfimorst    Validity checking mode
  -- -gnat y abcefhiklmnoprst Style checks

  package Compiler is
    for Default_Switches ("ada") use
      ("-gnatwcfilopru", "-gnatVcdfimorst", "-gnatyabcefhiklmnoprst");
  end Compiler;

  -- ld
  --
  -- -L      Library path (for libz.a)
  --         avoid space(s) and dot(s) in names, accept full qualified and relative paths
  -- -l      Library name (for libz.a)

  package Linker is

    -- valid full qualified path - .so shared lib first
    -- for Default_Switches ("ada")
    -- use ("-L/home/sr/Seafire/Sowebio/informatique/dev/ada/lib/zlib-1211", "-lz");
    -- valid relative path - .so shared lib first
    -- for Default_Switches ("ada") use ("-L../..", "-lz");

    -- valid relative path - specify libz.a static lib
    -- for Default_Switches ("ada") use ("-L../..", "-l:libz.a");

  end Linker;

  -- gprbuild
  --
  -- -s      Recompile if compiler switches have changed
  -- -gnatQ  Don't quit, write ali/tree file even if compile errors

  package Builder is
    for Default_Switches ("ada") use ("-s", "-gnatQ");
  end Builder;

end Zlib;
```

One can check that the program size has increased by about the same amount as the static library size. One can also check it visually with strace [the call to the library is pathless].

2.5 Statically linked executable embedding the run-time system

To statically link the runtime, you have to put the "-static" option in the binder and the linker, as in the AIDE build file below:

```
-- -----
-- 020
--
-- @file      aide.gpr
-- @copyright See authors list below and aide.copyrights file
-- @licence   GPL v3
-- @encoding  UTF-8
-- -----
-- @summary
-- aide library project file
--
-- @description
-- Build application and documentation
--
-- @authors
-- Stéphane Rivière - sr - sriviere@soweb.io
```

```

--
-- @versions
-- 20210317 - 0.1 - sr - initial release
-- 20210331 - 0.2 - sr - Add Style and GNATColl builds
-----

-- (0) invert comments for the 3 related lines to unlink gnatcoll sources
-- in order to generate pertinent documentation and true metrics

-- with "gnatcoll"; -- (0)
project aide is

  -- for Languages use ("Ada"); -- (0)
  for Languages use ("Ada", "C");

  type aide_Build_Type is ("Style", "Debug", "Fast", "Small");

  -- Add -Xaide_Build=Style in the GNATStudio build all window...
  -- %builder -Xaide_Build=Style -d %eL -P%PP %config %autoconf %X
  -- ...to directly control the build behaviour

  aide_Build: aide_Build_Type := external ("aide_Build", "Debug");

  -- for Source_Dirs use ("src/**", "../v20/src/**"); -- (0)
  for Source_Dirs use ("src/**", "../v20/src/**", "/home/sr/opt/gnat-2020/include/gnatcoll");

  case aide_Build is
    when "Style" =>
      for Object_Dir use "obj/style";
    when "Debug" =>
      for Object_Dir use "obj/debug";
      -- Use runtime with debug capabilities
      for Runtime ("Ada") use "/home/sr/opt/gnat-2020/lib/gcc/x86_64-pc-linux-gnu/9.3.1/rts-native-
debug";
    when "Fast" =>
      for Object_Dir use "obj/fast";
    when "Small" =>
      for Object_Dir use "obj/small";
  end case;

  for Exec_Dir use "bin";
  for Create_Missing_Dirs use "True";

  for Main use ("aide.adb");

  Common_Compiler_Options := (
    -- General
    "-gnatw8", -- Both brackets and UTF-8 encodings will be recognized (1)
    -- Warnings & Errors
    "-gnatu", -- Enable unique tag for error messages
    "-gnatf", -- Full errors. Verbose details, all undefined references
    "-gnatq", -- Don't quit, try semantics, even if parse errors
    "-gnatQ", -- Don't quit, write ali/tree file even if compile errors
    "-gnatVaep", -- Enable selected validity checking mode (2)
    "-gnatw.eDH.Y", -- Enable selected warning modes (3)
    -- "-Wall", -- Enable most warning messages
    -- Style
    "-gnatyaefhKM160npr" -- Enable selected style checks (4)
  );

  Style_Compiler_Options := (
    "-gnatg" -- RTS Style (6)
  );

  Debug_Compiler_Options := (
    "-gnata", -- Assertions enabled
    "-gnato", -- Enable overflow checking in STRICT mode
    "-gnateE", -- Generate extra information in exception messages
    "-gnateF", -- Check overflow on predefined Float types
    "-gnatVa", -- Enable all validity checking options
    "-fstack-check",
    "-fno-inline",
    --
    "-gnatec=" & project'Project_Dir & "aide.dbg",
    "-g" -- Generate debugging information
  );

  Fast_Compiler_Options := (
    "-O2",
    "-gnatpn",
    "-fipa-cp-clone", "-fgcse-after-reload",
    "-funroll-loops", "-fpeel-loops", "-funswitch-loops",
    "-ftracer", "-fweb", "-ftree-vectorize",
    "-frename-registers", "-ffunction-sections",
    "-g"
  );

```

```

Small_Compiler_Options := (
  "-Os"
);

-- (1)
-- https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gnat_ugn_unw/Character-Set-Control.html
-- https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gnat_ugn_unw/Wide-Character-Encodings.html#Wide-Charac-
ter-Encodings
-- (2)
-- a turn on all validity checking options
-- e turn on checking for elementary components
-- p turn on checking for parameters
-- (3)
-- .e turn on every optional info/warning (no exceptions)
-- D turn off warnings for implicit dereference (default)
-- H turn off warnings for hiding declarations (default)
-- .Y turn off info messages for why pkg body needed (default)
-- (4)
-- a check attribute casing
-- e check end/exit labels present
-- f check no form feeds/vertical tabs in source
-- h no horizontal tabs in source
-- k check casing rules for keywords
-- Mn check line length <= n characters
-- n check casing of package Standard identifiers
-- p check pragma casing
-- r check casing for identifier references
-- (5)
-- Options starting with -g, -f, -m, -O, -W, or --param are automatically passed on to the various
sub-processes
-- invoked by gcc. In order to pass other options on to these processes the -W<letter> options
must be used.
-- (6) All warnings and style messages are treated as errors. -gnatg implies -gnatw.ge and -gnatyg
so that all
-- standard warnings and all standard style options are turned on. All warnings and style messages
are treated
-- as errors.'

-- gnatmake options
package Compiler is
  case aide_Build is
    when "Style" =>
      for Default_Switches ("ada") use Common_Compiler_Options & Style_Compiler_Options;
    when "Debug" =>
      for Default_Switches ("ada") use Common_Compiler_Options & Debug_Compiler_Options;
      for Switches ("s-memory.adb") use ("-gnatg");
    when "Fast" =>
      for Default_Switches ("ada") use Common_Compiler_Options & Fast_Compiler_Options;
      for Switches ("s-memory.adb") use ("-gnatg");
    when "Small" =>
      for Default_Switches ("ada") use Common_Compiler_Options & Small_Compiler_Options;
      for Switches ("s-memory.adb") use ("-gnatg");
  end case;
end Compiler;

Common_Binder_Options := ("-static");

-- gnatbind options
package Binder is
  case aide_Build is
    when "Small" => for Default_Switches ("ada") use Common_Binder_Options;
    -- -Es: Store tracebacks in exception occurrences, and enable symbolic tracebacks
    when others => for Default_Switches ("ada") use Common_Binder_Options & ("-Es");
  end case;
end Binder;

Common_Linker_Options := ("-static");

-- ld options
package Linker is
  -- Static link with external C libs
  -- for Switches ("ada") use ("-L/home/sr/Seafire/Sowebio/informatique/dev/ada/lib/zlib-1211", "-
lz");
  case aide_Build is
    when "Style" =>
      for Default_Switches ("ada") use Common_Linker_Options;
    when "Debug" =>
      for Default_Switches ("ada") use Common_Linker_Options & ("-g");
    when "Fast" =>
      for Default_Switches ("ada") use Common_Linker_Options & ("-g", "-Wl,--gc-sections");
    when "Small" =>
      for Default_Switches ("ada") use Common_Linker_Options & ("-Wl,--gc-sections");
  end case;
end Linker;

-- gprbuild options
package Builder is

```

```

-- -d    Display compilation process
-- -j0   Use num processes to compile 0=all platform cores are used
-- -s    Recompile if compiler switches have changed
for Default_Switches ("ada") use ("-d", "-j0", "-s");
end Builder;

-- gnatdoc options
package Documentation is -- gnatdoc options
  for Documentation_Dir use "doc-generated";
end Documentation;

-- gnatpp option
package Pretty_Printer is
  for Default_Switches ("ada") use ("-M120", "-w8", "--comments-unchanged");
end Pretty_Printer;

-- gps options (to be reworked with appropriate options)
-- package Ide is
--   for Default_Switches ("adacontrol") use ("-f", "aide.aru", "-r");
-- end Ide;

-----
end aide;
-----

```

<https://this-page-intentionally-left-blank.org>

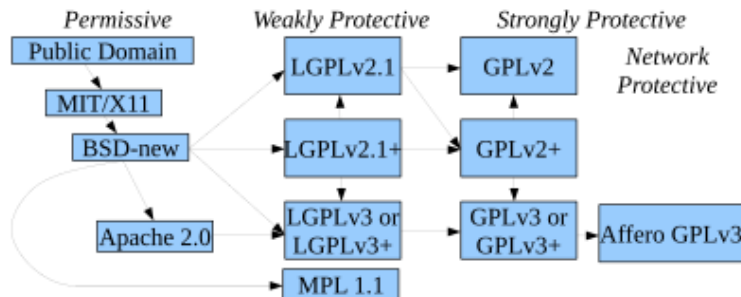


Appendices

1 Copyrights & credits

1.1 Library Licence

v20 is copyright Sowebio under GPL v3 license.



1.1.1 GPL v3 compatibility with others licenses

https://en.wikipedia.org/wiki/License_compatibility: MIT licence is compatible with GPL and can be re-licensed as GPL. European Union Public Licence [EUPL] is *explicitly compatible* with GPL v2 v3, OSL v2.1 v 3, CPL v1, EPL v1, CeCILL v2 v2.1, MPL v2, LGPL v2.1 v3, LiLIQ R R+ AGPL v3.

1.2 Manual license

Copyright © 2023 Stéphane Rivière. This document may be copied, in whole or in part, in any form or by any means, as is or with alterations, provided that alterations are clearly marked as alterations and this copyright notice is included unmodified in any copy.

2 To-do list Documentation

Hunt <<<TODO>>> tags :)

3 To-do list Software

3.1 Erroneous message after exception handling

4 Links

4.1 Ada

https://en.wikibooks.org/wiki/Ada_Programming

4.2 Others

Avr Freaks : <http://www.avrfreaks.net>

Adacore Github : <https://github.com/AdaCore>

Adacore Papers : <https://www.adacore.com/papers>

Adacore Gems : <https://www.adacore.com/gems>
Adacore Books : <https://www.adacore.com/books>
Adacore GPL : <https://www.adacore.com/community>

<http://www.tldp.org/HOWTO/Avr-Microcontrollers-in-Linux-Howto/x207.html>

<http://www.avrfreaks.net/forum/i-didnt-know-you-could-get-ada-avr>

4.3 People

4.3.1 Ludovic Brenta

Ada Debian Maintainer, and a good friend too.

<https://people.debian.org/~lbrenta/debian-ada-policy.html>

4.3.2 Stéphane Carrez

Member of Ada-France.

Blog: <https://blog.vacs.fr>

Sources repository: <https://github.com/stcarrez>

4.3.3 Frédéric Praca

Member of Ada-France.

Blog: <http://frederic.praca.free.fr>

4.3.4 Gautier de Montmollin

Prolific Ada program author [HAC, Lea, Azip, Gwindows, TexCad, among others], and a good friend too.

Blog: <https://gautiersblog.blogspot.com>

Sources repository: <https://github.com/zertovitch>



Ada, « it's stronger than you ».
Tribute to Daniel Feneuille, a legendary french Ada teacher [and much more]²⁰



²⁰ <http://d.feneuille.free.fr>