# ADA USER JOURNAL

Volume 43
Number 4
December 2022

## Contents

# Quarterly News Digest

***Alejandro R. Mosteo***

*Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain; email: amosteo@unizar.es*

## Contents

[Messages without Subject:/Newsgroups: are replies from the same thread. Messages may have been edited for minor proofreading fixes. Quotations are trimmed where deemed too broad. Sender's signatures are omitted as a general rule. —arm]

## Preface by the News Editor

Dear Reader,

Do not miss the upcoming Ada-Europe 2023 conference, for which an announcement can be found in this issue [1]. The conference will be celebrated, as always, in mid-June and will take place this year in lovely Lisbon.

I will also take this opportunity to congratulate Fabien Chouteau on receiving the 2022 ACM SIGAda Award for Outstanding Ada Community Contributions [2]. This award is entirely deserved, and I look forward to seeing his future initiatives to promote the Ada language in the open source community.

Finally, with this issue, we are mostly caught up and back on track with our regular news schedule. Remember that the last News Digest included only timely announcements, as the issue devoted most of its space to technical papers.

[1] "CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies", in Ada-related Events.

[2] "Winners of 2022 ACM SIGAda Awards", in Ada-related Events.

Sincerely,
Alejandro R. Mosteo.

## Ada-related Events

### ACM SIGAda HILT'22 Workshop on Supporting Rigorous S/W Development

[Event in the past, for the record. —arm]

*From: Tucker Taft <tucker.taft@gmail.com>*
*Subject: Re: ACM SIGAda HILT'22 Workshop on Supporting Rigorous S/W Development -- Oct 14, 2022*
*Date: Tue, 4 Oct 2022 13:59:44 -0700*
*Newsgroups: comp.lang.ada*

There is now an online option for attending the ACM SIGAda HILT'22 workshop featuring Niko Matsakis and Rustan Leino.

Anyone registered for the workshop will receive a link allowing use of Zoom and/or the "Whova" app to attend the workshop remotely. The organizers of the associated conference (ASE'22) have indicated that remote attendees may register at the lowest attendee price ("Student Member"). So if you or a colleague might be interested in participating in the workshop remotely, please register soon for the October 14th workshop, at: https://conf.researchr.org/attending/ase-2022/registration and indicate "Student Member" as your category of attendee.

[Original announcement omitted. —arm]

For more information see: https://conf.researchr.org/track/ase-2022/ase-2022-workshop-hilt-22

#formalmethods #softwareengineering #ada #rust #spark #dafny #ACM #ASE

### FOSDEM 2023: Call for Devroom

*From: Mockturtle <framefritti@gmail.com>*
*Subject: FOSDEM 2023: call for devroom. Deadline: 18/10*
*Date: Sun, 9 Oct 2022 10:11:14 -0700*
*Newsgroups: comp.lang.ada*

Dear all,
I just discovered that on 29/9 FOSDEM 2023 published the Call for DevRoom.

The deadline for the proposal is 18/10.

https://fosdem.org/2023/news/2022-09-29-call_for_devrooms/

*From: Dirk Craeynest <dirk.craeynest@gmail.com>*
*Date: Sun, 9 Oct 2022 23:17:28 -0700*

> I just discovered that on 29/9 FOSDEM 2023 published the Call for DevRoom.

We've been working behind the scenes on this already. Stay tuned for an announcement with more details on the AdaFOSDEM mailing list in the very near future!

Fer and Dirk

### Winners of 2022 ACM SIGAda Awards

*From: Tucker Taft <tucker.taft@gmail.com>*
*Subject: ANN: Winners of 2022 ACM SIGAda Awards*
*Date: Fri, 21 Oct 2022 11:54:27 -0700*
*Newsgroups: comp.lang.ada*

ACM Special Interest Group on Ada (SIGAda) is pleased to announce the following SIGAda awards for 2022.

==========
Winner of the 2022 Robert Dewar Award for Outstanding Ada Community Contributions, for broad, lasting contributions to Ada technology and usage:

Fabien Chouteau

Fabien Chouteau has been the lead of AdaCore's Ada Community outreach activities for many years. He has been the energy behind the "Make With Ada" and "Crate of the Year" contests, and has invigorated the Ada hobbyist market by encouraging support of amateur Ada champions, fostering the development of the excellent Alire package manager for Ada, and working to move all AdaCore libraries from GPL to a more permissive ("Apache 2.0") license.

==========
Winner of the 2022 ACM SIGAda Distinguished Service Award, for exceptional contributions to SIGAda activities and products:

Luis Miguel Pinho

Luis Miguel Pinho (PhD SMIEEE SMACM) is a Professor and Researcher in the Computer Engineering Department of the Polytechnic of Porto - School of

Engineering (ISEP), in Portugal. Miguel is the current editor of Ada Letters and serves as the SIGAda Secretary-Treasurer. He was a member of the Research Center in Real-Time and Embedded Computing Systems, and Executive Director of the Porto Research, Technology & Innovation Center. He served as General Chair and Program Co-Chair of Ada-Europe 2006 and General Co-Chair of ARCS 2015, was a Keynote Speaker at RTCSA 2010 and Program Co-Chair of Ada-Europe 2012, Ada-Europe 2016 and RTNS 2016. He was Editor-in-Chief of the Ada User Journal, and is a member of the HiPEAC network of excellence.

*From: Fabien Chouteau <fabien.chouteau@gmail.com>*
*Date: Wed, 26 Oct 2022 02:39:03 -0700*

Thanks a lot Tuck,
I am honored to receive the ACM SigAda award for my contribution to the Ada community.

It's been a blast working towards the broader adoption of Ada/SPARK, improving the ecosystem, and seeing the community evolve along the years. And this is a good opportunity for me to thank everyone who contributed to this effort, and/or who believed in our vision for the future of the Ada/SPARK community.

There is still a lot to do obviously, and I think the biggest challenge is to show those who have already seen Ada in the past how the language and its ecosystem have evolved. But we entered an exciting time for Ada/SPARK, as more and more people are questioning their choice of programming languages.

In my opinion, the most important topics for the future of Ada/SPARK are:

First, foster collaboration and welcome newcomers. This is why Alire and its ecosystem are game changing.

Second, spread awareness on the amazing power of SPARK, and have it recognized as the truly bleeding edge technology it is.

Third, use the technology to show what it can do. Since my first "Make With Ada" blog post in 2015, I have always been convinced that the best way to advocate for a technology is to use it. This is Make With Ada means to me.

Happy hacking!

## No Ada DevRoom in FOSDEM 2023

*From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>*
*Subject: No Ada DevRoom in FOSDEM 2023, alternative DevRooms and Ada-Europe support*
*Date: Tue, 8 Nov 2022 12:41:17 +0100*
*Newsgroups: comp.lang.ada*

Dear Ada community,

Our proposal for an Ada Developer Room for FOSDEM 2023 has been declined. I asked whether we could have a virtual DevRoom just like in FOSDEM 2022, but it seems unlikely. This means Ada will (most likely) not take part in the new edition of FOSDEM. We are saddened by this decision, but the amount of proposals was indeed very large: 88 DevRoom proposals were submitted!

Nonetheless, we would like to encourage Ada developers to submit presentations to other DevRooms that may fit your interests You can find the accepted DevRooms in [1]. I think the rooms that could be of interests to the Ada community are "Confidential Computing", "Embedded, Mobile and Automotive", "FOSS Educational Programming Languages", "Microkernel and Component-based OS", "Open Source Firmware, BMC and Bootloader" and "Security". However, take a look at all the proposals! Maybe you are writing some RISC-V or networking software in Ada, and there is a DevRoom just for it Please keep the AdaFOSDEM mailing list [2] informed about submissions and definitely about accepted proposals: we'll build a consolidated list of Ada-related talks at FOSDEM 2023, as we did before [3]. If you have any questions or issues, we will gladly help you where we can.

We are also happy to announce that Ada-Europe [4], after learning that there would be no Ada DevRoom in FOSDEM, has opened the possibility of adding a new "DevRoom like" track in their 2023 conference [5]. The Ada-Europe conference will take place in Lisbon between the 13 and 16 of June, 2023. If you are interested in this possibility, please, contact Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> to let him know.

Best regards,
The Ada FOSDEM team

[1] https://fosdem.org/2023/news/2022-11-07-accepted-developer-rooms/

[2] http://listserv.cc.kuleuven.be/archives/adafosdem.html

[3] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/21/210206-fosdem.html

[4] http://www.ada-europe.org/

[5] http://www.ada-europe.org/conference2023/cfp.html

## Advent of Code 2022

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Advent of Code 2022*
*Date: Sun, 4 Dec 2022 03:50:09 -0800*
*Newsgroups: comp.lang.ada*

In case you've missed it:
https://adventofcode.com/

There is even a chat room for Adaists about it @ https://forum.ada-lang.io/

Enjoy!

## Happy Birthday, Ada!

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Subject: Happy Birthday, Ada!*
*Date: Sat, 10 Dec 2022 11:35:20 +0100*
*Newsgroups: comp.lang.ada*

Born this date in 1815/1980.

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Sat, 10 Dec 2022 03:31:23 -0800*

Congratulation on Your Birthday,
Lady Ada

https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada Magica/20.html

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Sat, 17 Dec 2022 05:47:10 -0600*

Also the 10th Anniversary of Ada 2012.

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Sun, 18 Dec 2022 04:35:38 -0800*

> Also the 10th Anniversary of Ada 2012.
- Randy.

I would have bet that this date would be the release of ISO 2022. So it's going to be 2023?

## CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Subject: CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies*
*Date: Tue, 20 Dec 2022 16:49:01 -0000*
*Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc*

[CfC is included in the Forthcoming Events Section. —arm]

## Post-Ada Workshop at Ada-Europe 2023

*From: Marius Amado-Alves <amado.alves@gmail.com>*
*Subject: Post-Ada at Ada-Europe 2023 anyone?*
*Date: Wed, 21 Dec 2022 09:25:22 -0800*
*Newsgroups: comp.lang.ada*

Would anyone be interested in co-organizing or attending a Post-Ada workshop at Ada-Europe 2023 (Lisbon, 13-16 June)?

Any thoughts appreciated.

The "Post-Ada" concept has been debated here in CLA. It encompasses lessons learnt from the three decade long Ada experiment, ideas for betterment of the language, and creation of languages anew, like Parasail and King.

One way to approach the 'problem' would be to classify features of Ada as "keep, kill, or to be improved," for example:

- loop statements: keep
- function expressions: keep
- cursors: kill
- attributes vs. operations (tick vs. dot): kill
- inheritance: to be improved
- Unicode characters and strings: to be improved

A general issue could be to compare or harmonize this approach with the future (?) revision of Ada via Ada Issues. Personally I feel Ada (202X) is already too big to grow anymore. I suspect compiler maintainers would agree, and hope they could participate (sponsor?)

Maybe a full-day workshop with the structure:

1. plenary: presentations, debate coffee break
2. creation of a list of topics, of some kind of organization lunch
3. parallel sessions by subgroups of participants, by topic

coffee break

4. plenary: subgroup reports, debate, integration, conclusion, maybe plans for the future

Please relay at will.

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Wed, 21 Dec 2022 18:08:18 +0000*

> Would anyone be interested in co-organizing or attending a Post-Ada workshop at Ada-Europe 2023 (Lisbon, 13-16 June)?

> Any thoughts appreciated.

I probably won't be able to attend; my life is pretty much being destroyed right now.

> The "Post-Ada" concept has been debated here in CLA. It encompasses lessons learnt from the three decade long Ada experiment, ideas for betterment of the language, and creation of languages anew, like Parasail and King.

Really? No love for my "mad" :) ramblings?

https://github.com/Lucretia/orenda

> One way to approach the 'problem' would be to classify features of Ada as "keep, kill, or to be improved," for example:

> - loop statements: keep

> - function expressions: keep

> - cursors: kill

> - attributes vs. operations (tick vs. dot): kill

Wrong. Attributes are a really interesting and useful part of Ada and the solution in Orenda to getting addresses of objects, aspects would enable setting them on creation.

> - inheritance: to be improved

> - Unicode characters and strings: to be improved

Should be the basis of all text.

> A general issue could be to compare or harmonize this approach with the future (?) revision of Ada via Ada Issues. Personally I feel Ada (202X) is already too big to grow anymore. I suspect compiler maintainers would agree, and hope they could participate (sponsor?)

Won't happen, I've mentioned it before and was told it was not going to happen.

# Ada-related Resources

[Delta counts are from November 13th to February 12th. —arm]

## Ada on Social Media

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Ada on Social Media*
*Date: 12 Feb 2023 12:44 CET*
*To: Ada User Journal readership*

Ada groups on various social media:

| | |
|---|---|
| - Reddit: 8_291 (+91) members | [1] |
| - LinkedIn: 3_418 (+19) members | [2] |
| - Stack Overflow: 2_309 (+36) questions | [3] |
| - Telegram: 159 (+6) users | [4] |
| - Gitter: 151 (+11) people | [5] |
| - Ada-lang.io: 101 (+51) users | [6] |
| - Libera.Chat: 82 (+5) concurrent users | [7] |
| - Twitter: 32 (-5) tweeters | [8] |
| 49 (-36) unique tweets | [8] |

[1] http://www.reddit.com/r/ada/

[2] https://www.linkedin.com/groups/114211/

[3] http://stackoverflow.com/questions/tagged/ada

[4] https://t.me/ada_lang

[5] https://gitter.im/ada-lang

[6] https://forum.ada-lang.io/u

[7] https://netsplit.de/channels/details.php?room=%23ada&net=Libera.Chat

[8] http://bit.ly/adalang-twitter

## Repositories of Open Source Software

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Repositories of Open Source software*
*Date: 12 Feb 2023 12:44 CET*
*To: Ada User Journal readership*

| | |
|---|---|
| Rosetta Code: 920 (+1) examples | [1] |
| 39 (=) developers | [2] |
| GitHub: 763* (=) developers | [3] |
| Alire: 324 (+15) crates | [4] |
| Sourceforge: 240 (+2) projects | [5] |
| Open Hub: 214 (=) projects | [6] |
| Codelabs: 54 (+1) repositories | [7] |
| Bitbucket: 31 (=) repositories | [8] |
| AdaForge: 0** (-8) repositories | [9] |

*This number is unreliable due to GitHub search limitations.

**This site is currently unreachable.

[1] http://rosettacode.org/wiki/Category:Ada

[2] http://rosettacode.org/wiki/Category:Ada_User

[3] https://github.com/search?q=language%3AAda&type=Users

[4] https://alire.ada.dev/crates.html

[5] https://sourceforge.net/directory/language:ada/

[6] https://www.openhub.net/tags?names=ada

[7] https://git.codelabs.ch/?a=project_index

[8] https://bitbucket.org/repo/all?name=ada&language=ada

[9] http://forge.ada-ru.org/adaforge

## Language Popularity Rankings

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Ada in language popularity rankings*
*Date: 12 Feb 2023 12:44 CET*
*To: Ada User Journal readership*

[Positive ranking changes mean to go up in the ranking. —arm]

| | |
|---|---|
| - TIOBE Index: 23 (+4) 0.60% (+0.12%) | [1] |
| - PYPL Index: 17 (=) 0.94% (+0.13%) | [2] |
| - IEEE Spectrum* (general): 35 (=) Score: 1.16 | [3] |
| - IEEE Spectrum (jobs): 33 (=) Score: 0.79 | [3] |

- IEEE Spectrum (trending): 32 (=)
  Score: 3.95 [3]

*The Spectrum ranking has been revamped, no longer using the same categories and rating methodology. Thus, historic trends are omitted for this issue except for the default category.

[1] https://www.tiobe.com/tiobe-index/

[2] http://pypl.github.io/PYPL.html

[3] https://spectrum.ieee.org/top-programming-languages/

## XMPP Public Ada MUCs

*From: Alastair Hogge <agh@riseup.net>*
*Subject: Re: XMPP public Ada MUCs*
*Date: Wed, 7 Dec 2022 09:02:20 -0000*
*Newsgroups: comp.lang.ada*

Someone has created an Ada MUC [multi-user chat] at xmpp:ada@conference.magicbroccoli.de. It is low traffic at the moment.

Interested participates can sign up for free XMPP accounts at:

https://404.city/

https://magicbroccoli.de/register/

Some information on getting started with XMPP:

https://xmpp.org/getting-started/

## New Process for Submitting Comments about the Ada Language

*From: Tucker Taft <tucker.taft@gmail.com>*
*Subject: New process for submitting comments about the Ada language*
*Date: Sun, 18 Dec 2022 16:54:55 -0800*
*Newsgroups: comp.lang.ada*

[The announcement with the new commenting process for the Ada language standard appears in page 220 of this same issue —arm]

# Ada-related Tools

## AdaStudio-2022 Release 01/10/2022 Free Edition

*From: Leonid Dulman <leonid.dulman@gmail.com>*
*Subject: Announce: AdaStudio-2022 release 01/10/2022 free edition*
*Date: Sat, 1 Oct 2022 00:19:44 -0700*
*Newsgroups: comp.lang.ada*

I'm pleased to announce AdaStudio-2022.

It’s based on Qt-6.4.0-everywhere opensource (expanded with modules from Qt-5.15: qtgraphicaleffects qtgamepad qtx11extras qtwinextras), VTK-9.2.0, FFMPEG-5.1.1, OpenCV-4.6.0, SDL2-2.24.0, MDK-SDK(wang-bin)

Qt6ada version 6.4.0 open source and qt6base.dll ,qt6ext.dll (win64), libqt6base.so, libqt6txt.so(x86-64) built with Microsoft Visual Studio 2022 x64 Windows, GCC amd64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux amd64 Debian 11.2

AdaStudio-2022 includes the following modules: qt6ada, vtkada, qt6mdkada, qt6cvada (face recognition, QRcode detector, BARcode detection and others) and voice recognizer.

Qt6Ada is built under GNU LGPLv3 license https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada modules for Windows, Linux (Unix) are available from Google drive https://drive.google.com/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

WebPage is https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/index.html

[Removed detailed file contents. —arm]

The full list of released classes is in "Qt6 classes to Qt6Ada packages relation table.pdf"

The simple manual how to build Qt6Ada application can be read in "How to use Qt6ada.pdf"

## HAC v.0.21

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Ann: HAC v.0.21*
*Date: Sat, 1 Oct 2022 02:37:27 -0700*
*Newsgroups: comp.lang.ada*

HAC (HAC Ada Compiler) is a quick, small, open-source Ada compiler, covering a subset of the Ada language.

HAC is itself fully programmed in Ada.

Web site: http://hacadacompiler.sf.net/

From there, links to sources, and an executable for Windows.

Source repositories:

#1 svn: https://sf.net/p/hacadacompiler/code/HEAD/tree/trunk/

#2 git: https://github.com/zertovitch/hac

HAC is also available through Alire: https://alire.ada.dev/

* Main improvements since v.0.2:

- Added Virtual Machine Variables, another means for exchanging data between the HAC program and the program hosting the VM.
- SmallAda's tasking is working again in its HAC reincarnation -- at least, for some simple tasks.
- HAL becomes HAT (HAC Ada Toolbox), to avoid name collision with HAL = "Hardware Abstraction Layer".

Enjoy!

## LEA v.0.82

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Ann: LEA v.0.82*
*Date: Sat, 1 Oct 2022 02:46:10 -0700*
*Newsgroups: comp.lang.ada*

LEA is a Lightweight Editor for Ada

Web site: http://l-e-a.sf.net/

Source repository #1: https://sf.net/p/l-e-a/code/HEAD/tree/

Source repository #2: https://github.com/zertovitch/lea

Improvements:

- more ready-to-use Ada code samples
- improved Dark Side look
- indentation lines
- improvements in navigation (find/replace, compilation errors)
- embeds HAC v.0.21; details: see other post...

Features:

- multi-document
- multiple undo's & redo's
- multi-line edit, rectangular selections
- color themes, easy to switch
- duplication of lines and selections
- syntax highlighting
- parenthesis matching
- bookmarks

Currently available on Windows.

Gtk or other implementations are possible: the LEA_Common[.*] packages are pure Ada, as well as HAC.

Enjoy!

## VIM Bundle for Ada

*From: Martin Krischik <martin.krischik@gmail.com>*
*Subject: VIM bundle for Ada*
*Date: Tue, 11 Oct 2022 10:19:57 -0700*
*Newsgroups: comp.lang.ada*

I have updated the VIM bundle for Ada. If you are using VIM you should consider updating:

https://github.com/krischik/vim-ada

## GCC 12.1.0 macOS Cross-compiler to arm-eabi

*From: Simon Wright <simon@pushface.org>*
*Subject: Ann: GCC 12.1.0 macOS cross-compiler to arm-eabi*
*Date: Sat, 15 Oct 2022 20:11:45 +0100*
*Newsgroups: comp.lang.ada*

Find the above at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-arm-eabi.

Built on Intel, also runs on Apple silicon under Rosetta.

Scroll down to the bottom of the page to find the installation package.

## VisualAda for Visual Studio 2022 Release 1.0.0

*From: Alex Gamper <alby.gamper@gmail.com>*
*Subject: ANN: VisualAda (Ada Integration for Visual Studio 2022) release 1.0.0*
*Date: Sat, 15 Oct 2022 15:06:20 -0700*
*Newsgroups: comp.lang.ada*

Dear Ada Community,

VisualAda version 1.0.0 for Visual Studio 2022 has been released.

This is the initial release for Visual Studio 2022 and is a port of the existing VisualAda version 1.3 for Visual Studio 2017/2019.

Please feel free to download the free plugin from the following URL: https://marketplace.visualstudio.com/items?itemName=AlexGamper.VisualAda-2022

## VIM Plugin Update

*From: Martin Krischik <martin.krischik@gmail.com>*
*Subject: Another update to the VIM plugin.*
*Date: Tue, 25 Oct 2022 09:42:06 -0700*
*Newsgroups: comp.lang.ada*

Since GPS support was dropped for macOS having proper Vim plugins for Ada has become kind of important again. I added Alire compiler support so a press of <F7> will compile again.

It's actually two updated:

https://github.com/krischik/vim-ada/releases/tag/v_5.1.0

https://github.com/krischik/vim-ada/releases/tag/v_5.2.0

Have fun.

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Wed, 26 Oct 2022 00:21:19 -0700*

Thanks Martin,

I also recommend using neovim instead of vim, because of the builtin LSP (language-server protocol) support. We can then independently install the Ada language server from AdaCore (https://github.com/AdaCore/ada_language_server), and with a small configuration step we now have full cross-references in Ada...

The main difficulty is loading the proper project file. I will likely write a small blog post on the subject, though I could simply post the config I have here if there's interest.

*From: Martin Krischik <martin.krischik@gmail.com>*
*Date: Wed, 26 Oct 2022 07:45:36 -0700*

Thanks for the heads up.

[...]

Nice, there is a macOS version. But I notice no dependencies to any GUI framework and when I did try it out there was indeed no GUI support. I'm actually using GVim — the Vim with the graphical user interface and I'm not going back to a Terminal based editor. Still good to know the option exists.

## Gnu Emacs Ada Mode 7.3.1

*From: Stephen Leake <stephen_leake@stephe-leake.org>*
*Subject: Gnu Emacs Ada mode 7.3.1 released*
*Date: Wed, 26 Oct 2022 06:29:58 -0700*
*Newsgroups: comp.lang.ada*

Gnu Emacs Ada mode 7.3.1 is now available in GNU ELPA; the beta version has been promoted to release.

ada-mode and wisi are now compatible with recent GNAT versions. The grammar is updated to the proposed Ada 2022 version.

Incremental parse is provided. It still has some bugs, so it is not enabled by default. To try it:
(setq-default wisi-incremental-parse-enable t).

Incremental parse often gets confused; to recover, use M-x wisi-reset-parser. That does a full parse of the entire buffer, which can be noticeably slow in large buffers.

See the NEWS files in ~/.emacs.d/elpa/ada-mode-7.3.1 and wisi-4.0.0, or at http://www.nongnu.org/ada-mode/, for more details.

The required Ada code requires a manual compile step, after the normal list-packages installation ('install.sh' is new in this release):

```
cd ~/.emacs.d/elpa/ada-mode-7.3.1
./build.sh
./install.sh
```

This requires AdaCore gnatcoll packages which you may not have installed; see ada-mode.info Installation for help in installing them.

## Gnu Emacs Ada Mode 8.0 Beta

*From: Stephen Leake <stephen_leake@stephe-leake.org>*
*Subject: Gnu Emacs Ada mode 8.0 beta released.*
*Date: Mon, 07 Nov 2022 16:12:29 -0800*
*Newsgroups: comp.lang.ada*

Gnu Emacs Ada mode 8.0 beta is now available in GNU ELPA devel for beta testing.

All Ada mode executables can now be built with Alire (https://alire.ada.dev/); this greatly simplifies that process.

gpr-query and gpr-mode are split out into separate GNU ELPA packages. You must install them separately (Emacs install-package doesn't support "recommended packages" like Debian does).

Ada mode can now be used with Eglot; this is controlled by new variables:

ada-face-backend - one of wisi, eglot, none
ada-xref-backend - one of GNAT, gpr_query, eglot, none
ada-indent-backend - one of wisi, eglot, none

The indent and face backends default to wisi if the wisi parser is found in PATH, to eglot if the Ada LSP server is found, and none otherwise. The xref backend also looks for the gpr_query executable in PATH.

The current AdaCore language server (23) support face but not indent. The current version of eglot (19) does not support face. So for now, eglot + ada_language_server only provides xref.

The AdaCore language server ada_language_server is installed with GNATStudio (which ada-mode will find by default), or can be built with Alire. If you build it with Alire, either put it in PATH, or set gnat-lsp-server-exec.

I have not tested ada-mode with lsp-mode. You can set ada-*-backend to 'other to experiment with that, or tree-sitter, or some other backend.

To access the beta version via Gnu ELPA, add the devel archive to package-archives:
(add-to-list 'package-archives (cons "gnu-devel" "https://elpa.gnu.org/devel/"))

Then M-x list-packages; the beta release shows as ada-mode version 8.0.3.0.20221106.55317, wisi version similarly.

Please report success and issues to the Emacs ada-mode mailing list https://lists.nongnu.org/mailman/listinfo/ada-mode-users.

The required Ada code requires a manual compile step, after the normal list-packages installation:

```
cd ~/.emacs.d/elpa/ada-mode-7.3beta*
./build.sh
./install.sh
```

If you have Alire installed, these scripts use it. Otherwise, this requires AdaCore gnatcoll packages which you may not have installed; see ada-mode.info Installation for help in installing them.

## Artificial Intelligence Libraries

*From: Marius Amado-Alves <amado.alves@gmail.com>*
*Subject: Re: Artificial Intelligence libraries in ADA*
*Date: Thu, 10 Nov 2022 09:58:27 -0800*
*Newsgroups: comp.lang.ada*

Resurrecting this 3-year old thread, see what happens:-)

I too need AI and Machine Learning libraries, and I am literally disgusted at the perspective of having to use Python or Go or C++ for this. Has anything come up in the last 3 years? Maybe a binding to TensorFlow?

I plan to use Carter's REM NN, and maybe Kasakov's fuzzy_ml, for some experiments, but at some point, I'll want, like Bjorn Ludin, *recurrent* architectures, probably LSTM (Long Short* Term Memory), as I want to segment and classify text.

(Jeff: can we somehow reengineer REM NN towards recurrency? Maybe by inserting recurrent layers?)

*Not a typo. The ML geniuses really say "long short"...

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Thu, 10 Nov 2022 20:10:00 +0100*

> (Jeff: can we somehow reenginer REM NN towards recurrency? Maybe by inserting recurrent layers?)

Probably best to discuss this off line. You can contact me by e-mail.

*From: Rod Kay <rodakay5@gmail.com>*
*Date: Fri, 11 Nov 2022 21:20:04 +1100*

> Also, Rod Kay (charlie on irc) did something re TF iirc.

I generated a thin binding to the TensorFlow C API via swig4ada around mid June. The binding has not been tested apart from a 'hello_TF' demo which simply calls the 'TF_Version' function and prints it.

I've been distracted by other projects since but as chance would have it, I've recently resumed work on swig4ada and TF will definitely be one of the top priorities re testing swig4ada.

I'll try to take another look at it this weekend and to get the TF binding onto github, if possible.

## GCC 12.2.0 for macOS (x86_64 and aarch64)

*From: Simon Wright <simon@pushface.org>*
*Subject: ANN: GCC 12.2.0 for macOS (x86_64 and aarch64)*
*Date: Sun, 20 Nov 2022 19:02:46 +0000*
*Newsgroups: comp.lang.ada*

Find GCC 12.2.0 & tools for Intel silicon (will run on Apple silicon under Rosetta) at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.2.0-x86_64

Built on High Sierra with Python 3.9 (because Apple has withdrawn 2.7 in Monterey).

Also, the same for Apple silicon, built on Ventura but I've done my best to make sure it'll run on Monterey, at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.2.0-aarch64

I've marked both as pre-release, but I'm especially interested (if anyone has some time on their hands) in a check of the aarch64 version on Monterey.

## XNAdaLib and GNATStudio 2022 Binaries for macOS Monterey

*From: Blady <p.p11@orange.fr>*
*Subject: [ANN] XNAdaLib and GNATStudio 2022 binaries for macOS Monterey.*
*Date: Sat, 26 Nov 2022 09:07:51 +0100*
*Newsgroups: comp.lang.ada*

This is XNAdaLib 2022 built on macOS 12.6 Monterey for Native Quartz with GNAT FSF 12.1 (github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-x86_64) including:

- GTKAda 22.2 (www.adacore.com/gtkada) with GTK+ 3.24.33 (www.gtk.org) complete,
- Glade 3.40.0 (glade.gnome.org),
- Florist mid-2022a (github.com/Blady-Com/florist),
- AdaCurses 6.3 (patch 20221105) (invisible-island.net/ncurses/ncurses-Ada95.html),
- Gate3 0.5d (sourceforge.net/projects/lorenz),
- Components 4.64 (www.dmitry-kazakov.de/ada/components.htm),
- AICWL 3.25 (www.dmitry-kazakov.de/ada/aicwl.htm),
- Zanyblue 1.4.0 (zanyblue.sourceforge.net),
- PragmARC mid-2022 (pragmada.x10hosting.com/pragmarc.htm),
- UXStrings 0.4.0 (github.com/Blady-Com/UXStrings) - NEW
- GNOGA 2.2 mid-2022 (www.gnoga.com),
- SparForte 2.5 (sparforte.com),
- HAC 0.21 (https://hacadacompiler.sourceforge.io)

Here is also GNATStudio 23.0wb as a standalone app for macOS 12.

See readme for important details. There could be some limitations that you might meet. Feel free to report them on MacAda list (http://hermes.gwu.edu/archives/gnat-osx.html). Any help will be really appreciated.

Both packages have been posted on Source Forge: https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2022-monterey

## Simple Components v4.65

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: ANN: Simple Components v4.65*
*Date: Sat, 26 Nov 2022 23:08:41 +0100*
*Newsgroups: comp.lang.ada*

The library provides implementations of smart pointers, directed graphs, sets, maps, B-trees, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, persistent storage, multiple connections server/client designing tools and protocols implementations.

http://www.dmitry-kazakov.de/ada/components.htm

Changes the previous version:

- Bug fix in HTTP server causing memory leaks in accumulated bodies when browser keeps connection on;
- Python bindings, backward compatibility to lower versions of Python 3, e.g. 3.8;

- Julia and Python bindings for OSX corrected.

## Units of Measurement for Ada v3.12

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: ANN: Units of measurement for Ada v3.12 (New SI prefixes)*
*Date: Sat, 26 Nov 2022 23:11:57 +0100*
*Newsgroups: comp.lang.ada*

The library provides measurement unit support for Ada.

http://www.dmitry-kazakov.de/ada/units.htm

Changes to the previous version:

- Added four new SI prefixes adopted by General Conference on Weights and Measures (CGPM) in November 2022.

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 30 Nov 2022 07:44:13 -0800*

This is a bit confusing. From

https://www.lne.fr/en/news/general-conference-weights-and-measures-2022:

to express quantities of digital information using orders of magnitude in excess of 1024, has been adopted.

Thus, four new prefixes have been introduced:

ronna pour 1027

ronto pour 10-27

quetta pour 1030

quecto pour 10-30

End quote.

Abbreviations?

[...]

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Wed, 30 Nov 2022 18:03:36 +0100*

> ronna pour 1027

> ronto pour 10-27

> quetta pour 1030

> quecto pour 10-30

> End quote.

>

> Abbreviations?

q, r, R, Q

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 30 Nov 2022 09:49:48 -0800*

> I'm wondering when these prefixes will turn up in https://physics.nist.gov/cuu/Units/prefixes.html.

You can find them here:

https://www.bipm.org/en/measurement-units/si-prefixes

## The PragmAda Reusable Components

*From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>*
*Subject: [Reminder] The PragmAda Reusable Components*
*Date: Thu, 1 Dec 2022 11:57:28 +0100*
*Newsgroups: comp.lang.ada*

The PragmARCs are a library of (mostly) useful Ada reusable components provided as source code under the GMGPL or BSD 3-Clause license at https://github.com/jrcarter/PragmARC.

This reminder will be posted about every six months so that newcomers become aware of the PragmARCs. I presume that those who want notification when the PragmARCs are updated have used Github's notification mechanism to receive them, so I no longer post update announcements. Anyone who wants to receive notifications without using Github's mechanism should contact me directly.

## GNAT 12 on FreeBSD

*From: Alastair Hogge <agh@riseup.net>*
*Subject: GNAT-12 on FreeBSD*
*Date: Mon, 12 Dec 2022 05:17:53 -0000*
*Newsgroups: comp.lang.ada*

Description: This is an Ada compiler, from GCC-12.

Since Ada support must be built by an Ada-capable compiler, only platforms for which a bootstrap compiler is available can build it.

It is based on release versions of the Free Software Foundation's GNU Compiler Collection. It uses the GCC Runtime Library Exception, so the resulting binaries have no licensing requirements. Binaries produced by the AUX compiler should be legally handled the same as binaries produced by any FSF compiler.

It offers continuous improvements to the Ada 2022 standard since GCC 11.

https://www.freshports.org/lang/gnat12/
https://cgit.freebsd.org/ports/tree/lang/gnat12

## laceOS: an Operating System Tailored for Ada Development

*From: Rod Kay <rodakay5@gmail.com>*
*Subject: Ann: 'laceOS' ~ An operating system tailored for Ada development.*
*Date: Sat, 17 Dec 2022 17:23:24 +1100*
*Newsgroups: comp.lang.ada*

After spending many years installing various operating systems and setting them up for Ada development, I thought I'd try to make a simple OS installer which contains all the configuration and packages I usually use.

I thought this might be useful to others, perhaps lecturers/students, hobbyists, newcomers to Ada or anyone wanting to experiment with the latest Ada features.

The installer is very simple, asking a few questions (several with defaults) and takes about 10 minutes to do the installation.

Here is the Github link for anyone interested ...

https://github.com/charlie5/laceOS

Feedback/critique/suggestions most welcome.

Regards.

P.S. The installer is written in Ada. :)

## Adare_Net v0.0.128

*From: Daniel Norte De Moraes <danielcheagle@tutanota.com>*
*Subject: ANN: Adare_Net v0.0.128*
*Date: Mon, 19 Dec 2022 20:08:53 -0000*
*Newsgroups: comp.lang.ada*

Adare_Net new version v0.0.128:

Better code,

Added a adare_net.pdf manual,

Full client and server examples in udp and tcp.

Adare_net from version v0.0.128 approaches its v.0.1.0 version!

Adare_Net is a small, portable and easy to use Ada network lib. It supports ipv4 ipv6 udp and tcp, Socket Synchronous I/O Multiplexing and can 'listen' with ipv6, too.

https://github.com/danieagle/adare-net

## SDLAda 2.5.5

*From: Luke A. Guest <laguest@archeia.com>*
*Subject: [COTY] SDLAda-2.5.5 submitted*
*Date: Sat, 31 Dec 2022 14:27:23 +0000*
*Newsgroups: comp.lang.ada*

Just to inform people that SDLAda isn't dead, yet, it's just dormant. I finally got around my issues with Alire and submitted the 2.5.5 crate.

https://github.com/AdaCore/Ada-SPARK-Crate-Of-The-Year/issues/22

## References to Publications

### NSA Guidance on Software Memory Safety

*From: Jerry <list_email@icloud.com>*
*Subject: NSA Releases Guidance on How to Protect Against Software Memory Safety Issues*
*Date: Thu, 10 Nov 2022 15:48:00 -0800*
*Newsgroups: comp.lang.ada*

"Examples of memory safe languages include C#, Go, Java®, Ruby™, Rust®, and Swift®."

https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/

https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

Didn't the U.S. government once sponsor the development of a memory-safe language? (eye-roll)

## Ada and Other Languages

### MS Going to Rust (and Linux Too)

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: MS going to rust (and Linux too)*
*Date: Sat, 24 Sep 2022 09:52:34 +0200*
*Newsgroups: comp.lang.ada*

Apparently, Microsoft does not want to use C/C++ anymore:
https://www.zdnet.com/article/programming-languages-its-time-to-stop-using-c-and-c-for-new-projects-says-microsoft-azure-cto
and going to Rust. No word about glorious VBA and illustrious C#, though. The best ever inventions of the computing era deserve no mention... (:-))

Ah, GC does not sit well with them, who might think? (:-))

BTW, it seems that the Linux kernel will rust as well...

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Sat, 24 Sep 2022 09:50:33 +0100*

> Apparently Microsoft does not want to use C/C++ anymore:

Yeah, they're 20 years behind, I came to that conclusion then.

Well, people and companies will follow like sheep.

>No word about glorious VBA and illustrious C#

They'll stay as they are but likely will move to being implemented in rust.

[...]

> BTW, it seems that the Linux kernel will rust as well...

There was conversation about using zig as well a while ago.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sat, 24 Sep 2022 11:13:07 +0200*

> They'll stay as they are but likely will move to being implemented in rust.

I bet MS-Rust gets written in QBasic... (:-))

> There was conversation about using zig as well a while ago.

This one is from Linus himself.

Anyway, as expected, since computing resources begin actively stagnating, damn, even a used rusted (no pun intended (:-)) 3 years old HDD is twice more expensive now, the SW industry slowly turns away from well established practices of not caring about performance, efficiency, quality etc. I wonder, who will first dare proclaim that Agile was trash... (:-))

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Sat, 24 Sep 2022 04:09:48 -0700*

Sounds like "U.S. Department of Defense going to Ada" :-) ...

*From: G.B. <bauhaus@notmyhomepage.invalid>*
*Date: Sat, 24 Sep 2022 13:41:24 +0200*

> I wonder, who will first dare proclaim that [xyz] was trash... (:-))

Won't it be the presenter to use [xyz] in an economically informed speech about a new trendy replacement that is already a thing.

Trash in systems obeys a universal law, familiar to every consultant. That it piles up, and while leading to stagnation, trash also creates opportunities

- for oblivion,

- for cleaning out and

- for rebuilding.

A fresh start.

As a starting point, Rust has the fine mechanisms that will facilitate turning the language into a generator of consumable goods, including itself. It is, therefore, economically viable. By design, Rust meets many a business demand, since it doesn't stop at just technical ideas, of which it inherits many.

Write a really good driver for Linux using Ada 2012 and do not use capital letters in the source text, at least where Linux doesn't. Be silent about the language. Can an Adaist do that, to save the language?

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sat, 24 Sep 2022 14:31:38 +0200*

> Write a really good driver for Linux using Ada 2012 [...]

The song remains the same. No, Python need not to have Linux drivers in order to be hugely popular, like the Herpes virus need not to be...

And it is not about Ada. It is about a potentially turning point as the SW developing process hits certain limits one ignored before. Selling hot air is a very respectable and profitable activity, but in this case the reality begins showing its ugly bigotry face. Though Ada could provide some answers, it is not in the game anyway. Nevertheless, things become interesting...

*From: Nasser M. Abbasi <nma@12000.org>*
*Date: Sat, 24 Sep 2022 07:46:46 -0500*

> BTW, it seems that Linux kernel will rust as well...

This is a link that talks about using rust in Linux kernel

"Linux embracing Rust will boost robotics community"

"Linus Torvalds mentioned that the Rust programming language would be used in the upcoming Linux 6.1 kernel"

<https://www.therobotreport.com/linux-embracing-rust-will-boost-robotics-community/>

What I do not understand is, why not Ada instead of Rust? I thought Ada was designed for embedded low level software.

Maybe it is just more verbose than rust, and do not use {}.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sat, 24 Sep 2022 15:36:07 +0200*

> What I do not understand is, why not Ada instead of Rust?

Look at it this way. If Linus was not aware 30 years ago that there were better OSes than UNIX and better languages than C, why should he suddenly do now?

> Maybe it is just more verbose than Rust, and do not use {}.

It is never technical. You can try to rationalize your preference afterwards, but in reality, it is free will at play, even in the case of choosing Ada.

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Sat, 24 Sep 2022 10:29:54 -0700*

There is also a lot more emphasis on performance in the Rust world than in the Ada world. Part of this is due to resources, but a lot has to do with how the language itself is defined unfortunately. People working on the Linux kernel are definitely interested in performance (and remember they are using any programming language in a significantly different fashion than other programmers). People coming from C++ likely initially chose that language because it was advertised as the most performant.

*From: G.B.*
*<bauhaus@notmyhomepage.invalid>*
*Date: Sat, 24 Sep 2022 19:56:02 +0200*

> why should he suddenly do now?

Why not? He is actually talking about Rust, given C.

> It is never technical.

It needs to be technical to some extent. Suggesting to write a kernel in Python would encounter some technical opposition.

> You can try to rationalize your preference afterwards, but in reality, it is free will at play, even in the case of choosing Ada.

The point is that it's not free will. It seems about choice and about what drives choice. Some very old job descriptions very sincerely include "manipulating public opinion".

Think "Ada mandate"... Or better, don't, just don't.

*From: Dmitry A. Kazakov*
*<mailbox@dmitry-kazakov.de>*
*Date: Sat, 24 Sep 2022 21:07:01 +0200*

> Why not? He is actually talking about Rust, given C.

He is just growing old... (:-))

> It needs to be technical to some extent.

To some very infinitesimal extent. Actually my point was that the extent has an obvious tendency to grow now. Which is why we observe knee-jerk reactions from some weaklings... (:-))

> Suggesting to write a kernel in Python would encounter some technical opposition.

Honestly? The next generation will fully embrace Python as soon the last of the old farts retire. Linux held way too long, IMO... (:-))

> It seems about choice and about what drives choice.

Huh, in the not so distant future I expect drivers using HTTP to communicate inside the kernel encoding data in JSON and XML and written in JavaScript... I am almost serious. This garbage triumphally marches across embedded world right now, so no smiley.

## Ada Practice

### Reexposing Generics Formal Parameters

*From: Emmanuel Briot*
*<briot.emmanuel@gmail.com>*
*Subject: Calling inherited primitive operations in Ada*
*Date: Wed, 31 Aug 2022 01:15:38 -0700*
*Newsgroups: comp.lang.ada*

[Although the original post is about reusing inherited subprograms, the conversation quickly veered into a technical issue with generic formals, which is raised in the first answer. —arm]

A small blog post that you might find interesting:

https://deepbluecap.com/calling-inherited-primitive-operations-in-ada/

[...]

*From: Dmitry A. Kazakov*
*<mailbox@dmitry-kazakov.de>*
*Date: Wed, 31 Aug 2022 21:13:14 +0200*

This same technique is used in generics to work around another language design "feature":

```
generic
   type Foo is ...;
package
   subtype Actual_Foo is Foo;
```

*From: Emmanuel Briot*
*<briot.emmanuel@gmail.com>*
*Date: Wed, 31 Aug 2022 23:56:26 -0700*

To me, this is an orthogonal issue though (which would be worth its own blog post in fact). I can never remember (or perhaps not even understand) the reason for this limitation in Ada, which is a major pain when dealing with generics indeed…

I like the "Actual_" prefix, which I assume is some sort of convention in your code.

*From: amo...@unizar.es*
*<amosteo@unizar.es>*
*Date: Thu, 1 Sep 2022 00:57:33 -0700*

Is this about how according to some mystifying rules generic formals are[n't] visible from outside the generic?

*From: Dmitry A. Kazakov*
*<mailbox@dmitry-kazakov.de>*
*Date: Thu, 1 Sep 2022 12:02:43 +0200*

Right. I do not remember the rules, just the fact that they are quite logical. Unfortunately the logic of [it] is not very helpful. (:-))

As for primitive operations the problems are on many levels, from lacking introspection to missing inheritance of implementation by composition (AKA hooking).

*From: Jeffrey R.Carter*
*<spam.jrcarter.not@spam.acm.org.not>*
*Date: Thu, 1 Sep 2022 13:59:29 +0200:*

> Is this about how according to some mystifying rules generic formals are[n't] visible from outside the generic?

This seems like a non-issue to me. Any code that has visibility to a generic instance knows the actuals used for that instance. Can anyone provide real examples where this is a problem?

*From: Dmitry A. Kazakov*
*<mailbox@dmitry-kazakov.de>*
*Date: Thu, 1 Sep 2022 14:37:36 +0200*

> This seems like a non-issue to me. Any code that has visibility to a generic instance knows the actuals used for that instance.

That would make the code fragile. Should be avoided as much as possible as a form of aliasing. [...]

> Can anyone provide real examples where this is a problem?

Defaulted formal package actual part:

```
generic
   package Foo is new Bar (<>);
package Baz is ...
   -- What were these actuals in Foo?
```

This is one of most useful features used to reduce lists of formal parameters and simplify instantiations.

*From: Emmanuel Briot*
*<briot.emmanuel@gmail.com>*
*Date: Thu, 1 Sep 2022 07:10:03 -0700*

I have seen quite a number of cases of needing the subtype like Dmitry was showing. In a small number of cases, those were actual bugs in GNAT, but most of the time the compiler was correct. Mostly, you start to see the issue when you have generic packages that have formal generic packages.

Here is a quick example and the corresponding compiler error message. In Main, there is no way to see T. Of course, I can use Integer_Signature directly, but this is an issue.

If I rename Integer_Signature then I have to change a lot of places in my code (The aliasing that Dmitry was talking about)

```
with Signature;
generic
   with package Sign is new Signature (<>);
package Algo is
   procedure Compute (V : Sign.T) is null;
end Algo;

with Algo;
with Signature;
```

```
package Lib is
  package Integer_Signature is new
     Signature (Integer);
  package Integer_Algo is new Algo
    (Integer_Signature);
end Lib;

with Lib;
procedure Main is
  V : Lib.Integer_Algo.Sign.T;
  --  main.adb:3:24: "Sign" is not a visible
  -- entity of "Integer_Algo"
begin
  null;
end Main;

generic
  type T is private;
package Signature is
end Signature;
```

There are more interesting examples, somehow this one doesn't seem that bad. So here is another one:

```
generic
  type T is private;
package Gen is
end Gen;

with Gen;
generic
  type T is private;
  with package Must_Match is new
    Gen (T);
  with package Need_Not_Match is new
     Gen (<>);
package Gen2 is
  V1 : Must_Match.T;    --  "T" is not a
                -- visible entity of "Must_Match"
  V2 : Need_Not_Match.T;  -- instance of -
  -- same package,but this time T is visible
end Gen2;

with Gen, Gen2;
procedure P2 is
  package G is new Gen (Integer);
  package G2 is new Gen2
    (Integer, G, G);
begin
  null;
end P2;
```

I dug out the explanation that Tucker Taft once sent to the Ada-Comment mailing list (2019-11-14):

<<<

10/2

{AI95-00317-01} The visible part of a formal package includes the first list of basic_declarative_items of the package_specification. In addition, for each actual parameter that is not required to match, a copy of the declaration of the corresponding formal parameter of the template is included in the visible part of the formal package. If the copied declaration is for a formal type, copies of the implicit declarations of the primitive subprograms of the formal type are also included in the visible part of the formal package.

10.a/2

Ramification: {AI95-00317-01} If the formal_package_actual_part is (<>), then the declarations that occur immediately within the generic_formal_part of the template for the formal package are visible outside the formal package, and can be denoted by expanded names outside the formal package.If only some of the actual parameters are given by <>, then the declaration corresponding to those parameters (but not the others) are made visible.

10.b/3

Reason: {AI05-0005-1} We always want either the actuals or the formals of an instance to be nameable from outside, but never both. If both were nameable, one would get some funny anomalies since they denote the same entity, but, in the case of types at least, they might have different and inconsistent sets of primitive operators due to predefined operator “reemergence.” Formal derived types exacerbate the difference. We want the implicit declarations of the generic_formal_part as well as the explicit declarations, so we get operations on the formal types.

>>>

*From: amo...@unizar.es <amosteo@unizar.es>*
*Date: Thu, 1 Sep 2022 08:50:21 -0700*

> I have seen quite a number of cases of needing the subtype like Dmitry was showing. In a small number of cases, those were actual bugs in GNAT, but most of the time the compiler was correct.

> Mostly, you start to see the issue when you have generic packages that have formal generic packages.

This matches exactly my experience. I don't have enough grasp of the details to come up with a realistic short example, but I did hit this issue pretty often in two libs where I used signature packages quite extensively:

https://github.com/mosteo/rxada

https://github.com/mosteo/iterators

Initially I was always under the impression I was hitting GNAT bugs but then it turned out there were rules about it. A couple example places (you can see the renamings at the top. I was adding them "on demand" so to say):

https://github.com/mosteo/iterators/blob/master/src/iterators-traits-containers.ads

https://github.com/mosteo/rxada/blob/master/src/priv/rx-impl-transformers.ads

Thanks Emmanuel for the examples and digging out Tucker's explanation.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Thu, 1 Sep 2022 18:03:19 +0200*

> Mostly, you start to see the issue when you have generic packages that have formal generic packages.

None of these deal with the example I responded to

```
generic
   type T is ...
package P is
   subtype Actual_T is T;
```

> Reason: {AI05-0005-1} We always want either the actuals or the formals of an instance to be nameable from outside, but never both.

This is true in all these examples. I have used Ada since 1984, and this has never been a problem for me (as initially presented, this would have existed in Ada 83). Of course, I generally avoid generic formal packages. They seem to me to be a work around for poor design, and I prefer to correct the design.

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Thu, 1 Sep 2022 11:54:00 -0700*

I think I have a more interesting example. This one is extracted from my attempted traits containers, for which I had published a blog post at AdaCore. My enhanced fork of the library is at

https://github.com/briot/ada-traits-containers

if someone wants to experiment with non-trivial generics code (and containers are of course a case where generics fully make sense).

Here is the example:

```
generic
  type Element_Type is private;
  type Stored_Type is private;
package Elements is
end Elements;

with Elements;
generic
  type Element_Type is private;
package Definite_Elements is
  package Traits is new Elements
    (Element_Type, Stored_Type =>
     Element_Type);
end Definite_Elements;

with Definite_Elements;
generic
  type Key_Type is private;
package Maps is
  package Keys is new Definite_Elements
    (Key_Type);
  function "=" (L, R : Keys.
    Traits.Stored_Type) return Boolean
    --  "Stored_Type" is not a visible entity of
    -- "Traits"
```

```
   is (False);
end Maps;
```

This is not case where the actual is visible unless I happen to know how Definite_Element is implemented and that it will use Element_Type for Stored_Type (and this is not a knowledge I wish client packages to have, the whole point of Element and Definite_Element is to basically hide how elements can be stored in a container, and whether we need memory allocation for instance).

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Thu, 1 Sep 2022 23:33:44 +0200*

> [traits example]

As presented, this seems very over complicated. Elements and Definite_Elements add no value, and this is just a complex way of writing

```
generic
   type Key is private;
package Maps is
   function "=" (L : in Key; R : in Key) return
      Boolean is (False);
end Maps;
```

One hopes that the actual library makes better use of these packages than this small example. [...]

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Thu, 1 Sep 2022 23:11:45 -0700*

> Elements and Definite_Elements add no value, and this is just a complex way of writing

I did point you to the full repository if you prefer an extensive, real-life code sample. This was just an extract showing the gist of the issue.

> One hopes that the actual library makes better use of these packages than this small example.

[...] These packages are mostly implementation details. They are used to build high-level packages similar to the Ada containers, except with much better code reuse, more efficient, and SPARK-provable.

> Ada has excellent features for hiding, but these are not among them. Assuming

And that's exactly our point in this discussion. Ada on the whole is very good (we would not be using it otherwise), but it does have a number of limitations which are sometimes a pain, this being one of them. Not acknowledging the limitations when they exist is naive, all languages have them.

[...]

*From: amo...@unizar.es <amosteo@unizar.es>*
*Date: Fri, 2 Sep 2022 01:35:11 -0700*

> this seems very over complicated. Elements and Definite_Elements add no value, and this is just a complex way of writing

Going in a tangent, and I guess you know perfectly well, but this is caused by the painful duplication of code that Ada pushes you to by not having a native way to abstract storage of definite vs indefinite types. So the provider of a generic library very soon faces this conundrum about duplicating most interfaces, if not implementations, or resort to non-trivial generics, or accept an unnecessary penalty for definite types, or push to the client the definite storage matter. There's simply no satisfying solution here (that I know of). The duplication of every standard container to have both (in)definite variants is a strong indictment.

I can understand the desire to have full control of allocation and object sizes, but that there's not a language way to work around this duplication, with appropriate restrictions to go with it, is... bothersome. Not making a dig at the ARG, which I understand is overstretched as it is.

There was a proposal circulating some time ago that seemed promising, that I can't quickly find. Something like

```
type Blah is record
   Dont_care_if_in_heap: new
      Whatever_Definiteness;
  -- Would apply to indefinite types or formals
end record;
```

I don't think it made into https://github.com/AdaCore/ada-spark-rfcs/ or https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues or I can't find it.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 2 Sep 2022 10:48:53 +0200*

> I can understand the desire to have full control of allocation and object sizes, but that there's not a language way to work around this duplication, with appropriate restrictions to go with it, is... bothersome. Not making a dig at the ARG, which I understand is overstretched as it is.

Containers should be implementable without generics. Just saying.

> There was a proposal circulating some time ago [...]

I would prefer constraint propagation/management support + tuples instead:

```
   type Blah (Parameters :
      Whatever_Definiteness'Constraints) is
      Sill_Care : Whatever_Definiteness
         (Parameters);
   end record;
```

*From: amo...@unizar.es <amosteo@unizar.es>*
*Date: Fri, 2 Sep 2022 02:20:44 -0700*

> Containers should be implementable without generics. Just saying.

Are you now referring to current Ada or to hypothetical features?

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 2 Sep 2022 11:55:11 +0200*

> Are you now referring to current Ada or to hypothetical features?

Hypothetical like all types having classes and supertypes. E.g. when you instantiate generic with a type you semantically place the type in the implicit class of formal types of the generic. You cannot do that now without generics. Furthermore, there is no way to describe relationships between types like array index and array, like range and discrete type of its elements etc.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Fri, 2 Sep 2022 12:41:42 +0200*

> [...]not having a native way to abstract storage of definite vs indefinite types [...]

The only indefinite data structure that is needed seems to be holders. Any other indefinite data structure can be implemented as the equivalent definite data structure of holders, so there need be no duplication of implementations. That one cannot use a single pkg for both does result in duplication of the spec, but that seems like less of an issue to me.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 2 Sep 2022 13:04:34 +0200*

> The only indefinite data structure that is needed seems to be holders.

The language should support and encourage design that does not rely on memory pools.

In my view one of the major advantages of Ada is that indefinite objects can be handled without resorting to hidden or explicit pointers to pools.

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Fri, 2 Sep 2022 04:20:38 -0700*

> I was not willing to spend more than about 15 minutes trying to understand this, so I may be missing something.

Fair enough. The library is really a set of experiments, mostly successful, and I think it might have been used for the implementation of the current SPARK containers in GNAT, though I am not positive there. I did look at the pragmARC components, and there you indeed chose to have a large number of similar-looking packages and code duplication. I guess we'll have just to agree to disagree on the design approach there. But of course, users having choices is what makes an ecosystem interesting.

What I was really going after are graphs and their algorithms. In particular, I want those algorithms to work on any graph data structure provided it has a number of primitive operations. In fact, the algorithm could also work when the graph is kind of implicit in the code, even if we do not have an actual Graph object. And for this, you need generics.

A similar approach is what Rust uses all over the place with its traits, or what C++ Boost library uses for its graphs, too.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Fri, 2 Sep 2022 19:01:28 -0500*

>[...] painful duplication of code that Ada pushes you to by not having a native way to abstract storage of definite vs indefinite types.

This is premature optimization at its worst.

> [...]

There is no penalty in a code sharing implementation like Janus/Ada: the implementation of definite types is essentially the same as you would write by hand for an indefinite type. In most cases, all one needs is an indefinite generic.

(The plan for Janus/Ada was always to use post-compilation optimization to reduce the overhead of generics, but admittedly, that part never got built. If I had infinite time...)

Assuming otherwise is certainly premature optimization.

> There's simply no satisfying solution here [...]

The original expectation for the containers was that there would be many variants of each container, because the needs for memory management, task management, and persistence differ between applications: there is no one-size fits all solution.

But I agree on one point: the "basic" container is unnecessary; one should either use the indefinite or bounded container (depending on your memory management needs, either fully fixed or fully heap-based)

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Fri, 2 Sep 2022 19:07:27 -0500*

>These packages are mostly implementation details [...]

(Wading in where I should probably not tread... :-)

But they violate the #1 principle of the Ada.Containers: ease of use. One principle that we insisted on was that a single instantiation was the maximum we would use, because we did not want people moving from arrays to containers to have to replace one declaration with a half page of magic incantations. (This is the reason that there is no container interface, for one consequence, and certainly no signature packages.)

In general, people either understand and like signature packages, or really do not understand them and just use them when insisted on. The standard containers in Ada needed to be usable by the maximum number of users, and insisting on bells and whistles that many don't understand does not help.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Fri, 2 Sep 2022 19:12:25 -0500*

> In my view one of the major advantages of Ada is that indefinite objects can be handled without resorting to hidden or explicit pointers to pools.

But they're implemented with some sort of hidden allocation. (GNAT uses a "secondary stack", whatever that is, but that is just a restricted form of pool). Janus/Ada uses built-in pools with cleanup for all such things to simplify the interface (the code for allocations and stand-alone objects is mostly shared, both within the compiler and at runtime).

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sat, 3 Sep 2022 10:23:01 +0200*

> But they're implemented with some sort of hidden allocation. [...]

For a programmer that does not matter. The problem with pools is locking, non-determinism, issues with protected actions. If [the] secondary or primary stack is the program stack, nobody really cares.

BTW, merely doing pool tracing/bookkeeping becomes a sheer nightmare if you cannot return a string from a function.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Sat, 3 Sep 2022 10:59:16 +0200*

> One principle that we insisted on was that a single instantiation was the maximum we would use

Except for queues

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Tue, 6 Sep 2022 19:42:57 -0500*

> Except for queues

Right, and one consequence of that is that the queues aren't used much. (Not sure if they would be used much in any case, they're definitely a specialized need compared to a map.)

*From: Simon Wright <simon@pushface.org>*
*Date: Sat, 03 Sep 2022 20:00:00 +0100*

> One principle that we insisted on was that a single instantiation was the maximum

And this was one reason that I didn't put up any arguments at Ada Europe 2002 for the Ada 95 Booch Components to form a basis for Ada.Containers - you'd need 3 instantiations, one after the other.

```
--  A company's Fleet holds a number of
-- Cars.
  with BC.Containers.Collections.Bounded;
  with Cars;
  package My_Fleet is

    use type Cars.Car;

    package Abstract_Car_Containers
    is new BC.Containers (Cars.Car);

    package Abstract_Car_Collections
    is new
      Abstract_Car_Containers.Collections;

    package Fleets
    is new
      Abstract_Car_Collections.Bounded
      (Maximum_Size => 30);

    The_Fleet : Fleets.Collection;

  end My_Fleet;
```

The other was a lack of consistency in the implementation (Length? Size?).

*From: Emmanuel Briot <briot.emmanuel@gmail.com>*
*Date: Sun, 4 Sep 2022 23:56:43 -0700*

> for the Ada 95 Booch Components [...] you'd need 3 instantiations

I definitely see the same issue. The way my library is trying to workaround that is as follows: Those instantiations are only needed for people who want/need to control every aspect of their containers, for instance how elements are stored, how/when memory is allocated, what is the growth strategy for vectors, and so on.

Most users should not have to care about that in practice. So we use code generation at compile time to generate high-level packages similar to the Ada containers, with a limited set of formal parameters (in src/generated, to be more specific). We can generate bounded/unbounded versions, definite/indefinite versions, and any combination of those.

One of the intentions of the library, initially, had been the implementation of the Ada containers and SPARK containers in GNAT, as a way to share as much code as possible between the two.

Randy Brukardt:

> Assuming otherwise is certainly premature optimization.

I am quoting a bit out of context, though I believe it is close enough. Designers of containers must care about performance from the get-go. Otherwise, people might just as well use a list for everything and just traverse the list all the time. We all know this would be way too inefficient, of course, which is why there are various sorts of containers. Anyone who has actually written performance-sensitive code knows that memory allocations are definitely something to watch out for, and the library design should definitely take that into account.

Jeff Carter:

> The only indefinite data structure that is needed seems to be holders

Although it is certainly true that using holders works, it is not applicable when designing a containers library that intends to be mostly compatible with Ada containers. The latter have chosen, long ago and before Holder was a thing, to have definite and indefinite versions. The main benefit to this approach is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string. I must admit I have very limited experience with Holders, which I have never used in production code (nor, apparently, have my colleagues and ex-colleagues).

Randy Brukardt:

> Ada *DOES* support default values for formal parameters of generics

Hey, I just discovered that, thanks Randy ! For people who also did not know that:

```
generic
    type Item_Count is range <>  or use
      Natural;
package Gen is
```

It is supported by GNAT's newer versions (I don't know when it was implemented though)

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Mon, 5 Sep 2022 09:34:37 +0200*

> Although it is certainly true that using holders works, it is not applicable when designing a containers library that intends to be mostly compatible with Ada containers.

Right. Holder requires finalization and finalization means language prescribed finalization lists which is highly undesirable in many cases.

> The main benefit to this approach is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string.

And that the container designer has control over the pool where the items get actually allocated.

> I must admit I have very limited experience with Holders, which I have never used in production code (nor, apparently, have my colleagues and ex-colleagues).

I have been using the idea for a long time, since Ada 95 before the standard library had them. In my experience holders multiply the number of container variants:

1. Definite elements

2. Indefinite elements

+

3. Holder elements in the interface (and maybe implementation)

The third gets a holder package as a formal parameter or, alternatively, is a child of a holder package (for performance reasons). The container interface has direct operations in terms of the Element_Type as well as in terms of the holder type.

Sometimes the holder variant is actually the indefinite one that promotes holders only in its interface.

P.S. In my opinion helper types/package is an evil of far greater scale than any premature optimization!

The programmers doing the latter at least try to understand the code they write.

*From: amo...@unizar.es <amosteo@unizar.es>*
*Date: Mon, 5 Sep 2022 01:53:19 -0700*

> This is premature optimization at its worst.

Just because the language doesn't offer a way to do it. Otherwise I wouldn't need to care.

> There is no penalty in a code sharing implementation like Janus/Ada

Well, that sounds neat for Janus/Ada, but is a different issue to clients having to wrap their indefinite types prior to instantiation, and suffer the unwrapping throughout the code.

> Assuming otherwise is certainly premature optimization.

I'm of the opinion that it goes beyond just premature optimization, in the terrain of readability/maintainability by causing boilerplate, and when generics specializations do become necessary, by causing code duplication.

[...]

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 5 Sep 2022 11:30:56 +0200*

> Those instantiations are only needed for people who want/need to control every aspects of their containers [...] So we use code generation at compile time to generate high-level packages similar to the Ada containers

This seems backwards. The user should encounter the forms most likely to be used first; if part of the packages are in a subdirectory, those should be the ones less likely for the typical user to use.

> The main benefit to this approach [definite+indefinite containers] is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string.

Before Ada.Containers existed, I commonly used definite data structures from the PragmARCs with Unbounded_String, which is partly a specialized holder for String (and partly a specialized Vector for Positive/Character). Generalizing from this led to implementing a Holder pkg.

When I said a holder is the only indefinite pkg that is needed, I meant to imply that other indefinite structures would be implemented using the definite form + holder.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Tue, 6 Sep 2022 19:51:44 -0500*

> I am quoting a bit out of context

Definitely out of context. If you have code which is truly performance sensitive, then it cannot also be portable Ada code. That's because of the wide variety of implementation techniques, especially for generics. (For Janus/Ada, if you have critical performance needs, you have to avoid the use of generics in those critical paths -- sharing overhead is non-zero.)

I agree that the design of the containers matters (which is why we made the sets of operations for the various containers as close as possible, so switching containers is relatively easy). But the indefinite/definite thing is premature optimization - it makes little difference for a Janus/Ada generic, at least in the absence of the full-program optimizer (that we never built). If your code isn't intended to be portable Ada code, then *maybe* it makes sense to worry about such things. But the expectation was always that containers would be useful in cases where the performance is *not* critical - one probably should use a custom data structure for performance critical things. (But most things aren't really performance critical in reality.)

## GNAT Speed Comparison on Older Intel versus Apple Silicon M1

*From: Jerry <list_email@icloud.com>*
*Subject: GNAT Speed Comparison on Older Intel versus Apple Silicon M1*
*Date: Tue, 8 Nov 2022 20:07:32 -0800*
*Newsgroups: comp.lang.ada*

I use GNAT on a late 2008 MacBook Pro with a 2.4 GHz Intel Core 2 Duo for heavy numerical computing. It is not uncommon for my programs to run several minutes to several hours. Does anyone have a feel for how much speed increase I would see using GNAT on an Apple Silicon M1 PowerBook Pro? My main curiosity is single-core runs since GNAT does not parallelize; I am aware that I can run multiple programs simultaneously on multiple cores.

*From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>*
*Date: Wed, 9 Nov 2022 08:38:48 +0100*

Hi Jerry,

taking the results from Geekbench: [1] for your current MacBook and [2] for the M1 MacBook from 2021; the results show that single core performance of the M1 MacBook Pro is about 6.4 times faster.

However, notice that it is running on Aarch64 natively for the M1. Nonetheless, you can run x86 programs with little performance hit thanks to Apple Rosetta.

Also, GNAT afaik, allows for parallel computations using tasks. The multicore performance gain between the two models is about 24x.

These results are however just an average. Maybe your program does not see such improvements as it may bottleneck earlier or it may see greater gains.

Regards,

[1] https://browser.geekbench.com/macs/macbook-pro-early-2008

[2] https://browser.geekbench.com/v5/cpu/18518008

*From: Jerry <list_email@icloud.com>*
*Date: Wed, 9 Nov 2022 22:26:18 -0800*

> Hi Jerry,

> taking the results from Geekbench:

That's a great site. Thanks. Clicking around a bit I was able to find separate comparisons for single-core floating point and the speed-up is 5.2.

> However, notice that it is running on Aarch64 natively for the M1.

GNAT compiles to Aarch64 now, right?

> Nonetheless, you can run x86 programs with little performance hit thanks to Apple Rosetta.

"little performance hit" compared to Intel code running on Rosetta versus Intel silicon or compared to native ARM? And I wonder how long until Apple takes away Rosetta this time? Last time it was two OS updates and then, poof, gone.

*From: Simon Wright <simon@pushface.org>*
*Date: Sun, 13 Nov 2022 16:29:54 +0000*

> GNAT compiles to Aarch64 now, right?

You can download an aarch64-apple-darwin21 compiler for C, C++, Ada at [1]. However, it won't compiler C (or, I guess, C++) on Ventura - I'm working on a GCC 12.2 version.

[1] https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-aarch64-1

## Variable Value If Exception Is Raised

*From: Nytpu <alex@nytpu.com>*
*Subject: Variable value if exception is raised*
*Date: Sun, 20 Nov 2022 18:03:04 -0000*
*Newsgroups: comp.lang.ada*

Hello everyone,

If an exception is *explicitly* raised during a variable assignment, what happens to the variable contents Are they in an undefined ("abnormal") state, or are the previous contents preserved?

For example:

```
with Ada.Text_IO;
procedure Test is
   function Always_Raises return Integer is
   begin
      raise Program_Error;
      return 1;
   end Always_Raises;

   I : Integer := 0;
begin
   -- insert a nested handler, because the
  -- ARM § 11.4 ¶ 3 *does* say that the
  -- currently executing body is "abnormally
  -- completed" (including finalizing
  -- everything) before
   -- entering the exception handler
   begin
      I := Always_Raises;
   exception
      when others => null;
   end;
   Ada.Text_IO.Put_Line(Integer'Image(I));
end;
```

What, if anything, will be printed (Disclaimer: I know the preexisting variable value will be preserved in GNAT specifically, but I'm asking if the standard guarantees that's the case)

I read through the ARM 2012 § 11 and § 5.2, as well as skimming through everything related to “assignment” and “exceptions” in the ARM index; and didn't see much relating to this. All I saw is this:

> When an exception occurrence is raised by the execution of a given construct, the rest of the execution of that construct is abandoned

— ARM 2012 § 11.4 ¶ 3

Which I guess implicitly protects variable values since assigning to a variable is performed after evaluating the right hand side, but still not necessarily a clear answer.

I did see in § 13.9.1 that language-defined validity checks (e.g. bounds checks) failing or calling `abort` in a task during an assignment will cause the variable to enter an "abnormal" (i.e. invalid) state, but that doesn't cover user-raised exceptions.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Sun, 20 Nov 2022 20:00:32 +0100*

If the exception occurs during evaluation of the RHS, as in your example, then the language guarantees that the value of the LHS is unchanged. The execution of the assignment statement is abandoned before the value of the LHS is changed.

If an exception is raised while adjusting a controlled LHS, then the value of the LHS has already been changed before the exception is raised.

> -- insert a nested handler, because

> -- the ARM § 11.4 ¶ 3 *does*

> -- say that the currently executing

> -- body is "abnormally completed"

> -- (including finalizing everything)

> -- before entering the exception

> -- handler

This comment is false. Finalization does not occur until the exception handler finishes. Exception handlers would be pretty useless otherwise.

## String View of File

*From: Jesper Quorning <jesper.quorning@gmail.com>*
*Subject: String view of file*
*Date: Mon, 21 Nov 2022 00:30:00 -0800*
*Newsgroups: comp.lang.ada*

Is it possible to write something like this with Ada

```Ada
package my_rw_file is new file
  (name => "whatever",
   mode => read_write,
   implementation => standard
   -- or portable or fast
  );
package as_string is new xxx
   (from => my_rw_file);
-- parse (as_string);
package data is new parse (as_string,
format => markdown); -- or whatever
```

Sorry, I’m new to Ada

*From: G.B. <bauhaus@notmyhomepage.invalid>*
*Date: Mon, 21 Nov 2022 14:01:01 +0100*

Do you mean, gobble up a file into a string and then parse that? Yes, that's possible in a number of ways.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 21 Nov 2022 14:48:47 +0100*

[...]

If you want to read the arbitrary contents of a file into a String, that's easily done:

```
with Ada.Directories;
package String_A_File is
   use type Ada.Directories.File_Size;

   function File_As_String (Name : in String)
   return String with
     Pre  => Ada.Directories.Exists (Name)
   and then
       Ada.Directories.Size (Name) <=
       Ada.Directories.File_Size (Integer'Last),
       Post => File_As_String'Result'First = 1
   and
        File_As_String'Result'Last =
        Integer (Ada.Directories.Size (Name) );
end String_A_File;

with Ada.Sequential_IO;

package body String_A_File is
   function File_As_String (Name : in String)
return String is
      subtype FAS is String (1 .. Integer
      (Ada.Directories.Size (Name) ) );

      package FAS_IO is new
Ada.Sequential_IO (Element_Type => FAS);
      File   : FAS_IO.File_Type;
      Result : FAS;
   begin -- File_As_String
      FAS_IO.Open (File => File, Mode =>
        FAS_IO.In_File, Name => Name);
      FAS_IO.Read (File => File, Item =>
        Result;
      FAS_IO.Close (File => File);
      return Result;
   end File_As_String;
end String_A_File;
```

This presumes that Result will fit on the stack. If that's likely to be a problem, then you will need to use Unbounded_String and read the file Character by Character.

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Mon, 21 Nov 2022 17:52:14 +0200*

For the OP's benefit (Jeffrey of course knows this): an alternative to Unbounded_String is to allocate the Result string on the heap, and return an access to the heap string. With that method, you can still read the entire string with one call of FAS_IO.Read instead of Character by Character.

*From: Marius Amado-Alves <amado.alves@gmail.com>*
*Date: Mon, 21 Nov 2022 08:11:05 -0800*

Use Ada.Sequential_IO (Character), load to an Unbounded_String, save from a String or Unbounded_String.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 21 Nov 2022 17:42:56 +0100*

> For the OP's benefit (Jeffrey of course knows this)

I know it, and I deliberately reject it. Having access types in a pkg spec is poor design. Delegating the associated memory management and all its opportunities for error to the pkg client is very poor design.

If access types are used, they should be hidden and encapsulated with their memory management. This makes it easier to get the memory management correct. Since this is what using Unbounded_String does for you, I think it's better to use it than to expend extra effort doing something similar.

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Mon, 21 Nov 2022 19:29:12 +0200*

> I know it, and I deliberately reject it.

I agree in general, but there are design trade-offs that depend on issues not made clear in the original question, such as the size of the file and the performance requirements. So I thought that the OP should know of the heap alternative, even if it has some poorer properties too.

*From: Qunying <zhu.qunying@gmail.com>*
*Date: Mon, 21 Nov 2022 09:29:49 -0800*

If you are using GNAT with gnatcoll, then you may try its mmap facility, https://docs.adacore.com/gnatcoll-docs/mmap.html

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Mon, 21 Nov 2022 13:43:39 -0800*

You may be interested by this:

https://github.com/zertovitch/zip-ada/blob/master/zip_lib/zip_streams.ads#L148

It can be actually used out of the context of Zip archives.

## Ada 2022 in GNAT

*From: Simon Belmont <sbelmont700@gmail.com>*
*Subject: Ada 2022 in GNAT*
*Date: Thu, 8 Dec 2022 15:37:15 -0800*
*Newsgroups: comp.lang.ada*

Has anyone seen (or willing to type up...) any broad-strokes information about how GNAT (et al) actually plans to implement the parallelization features of Ada 2022? Take advantage of GPUs or just stick to CPU cores, or some kind of binding to OpenMP, etc, on Linux vs Windows vs Vworks, etc? I'm mostly just curious and haven't seen any of that in-the-weeds type information floating around, or at least anything that isn't a few years old.

*From: Fabien Chouteau <fabien.chouteau@gmail.com>*
*Date: Fri, 9 Dec 2022 09:07:50 -0800*

> In the past year or so, we have been working hard assessing and implementing most of these Ada 202x changes (called AIs: Ada Issues in ARG terms). The implementation work and feedback from first users allowed us to identify that a few of these features would need additional time and attention. This led us to make a difficult decision - in order to allow for more investigation and to avoid users to start to rely on constructs that may need to change or be replaced, we decided to put on hold the implementation of some of the changes in language. Of course, we’re currently engaged with the ARG to discuss these.

> The main set of features that AdaCore and GNAT are putting on hold are related to the support for parallel constructs. While the overall vision is an exciting and promising one, we realized when looking at the state of the art and gathering user requirements that there were a lot more aspects to consider on top of those currently addressed by the AIs. Some of these are related to GPGPU (General Purpose GPU) support as well as their future CPU counterparts, and include topics such as control of memory transfer, precise allocation of tasks and memory on the hardware layout, target-aware fine tuning options as well as various other parametrization needs. These capabilities happen to be fundamental to obtain actual performance benefits from parallel programming, and providing them may require profound changes in the language interface. Consequently, we’re putting all parallel AIs on hold, including support for the Global and Nonblocking aspects beyond the current support in SPARK.

See https://blog.adacore.com/ada-202x-support-in-gnat

*From: Simon Belmont <sbelmont700@gmail.com>*
*Date: Sat, 10 Dec 2022 12:03:08 -0800*

> See https://blog.adacore.com/ada-202x-support-in-gnat

That post is over two years old, surely that can't still be the state of things? I'm not sure what it says when the big, marquee item of the newest standard isn't even actively being worked on in the big, marquee compiler.

*From: Fabien Chouteau <fabien.chouteau@gmail.com>*
*Date: Tue, 13 Dec 2022 04:10:43 -0800*

> That post is over two years old, surely that can't still be the state of things?

There has been progress on the Ada2022 support, but no change on that part.