To receive the Ada User Journal, apply to Ada-Europe at:
http://www.ada-europe.org/**join**

ADA USER JOURNAL

Volume 42
Numbers 3-4
September – December 2021

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,

This period has been particularly rich with maintenance-related woes. See, for example, how the removal of features to ease maintenance burden can also be a problem for long-time users, in this case in an ASIS-related discussion with interesting points about the differences in LibAdalang philosophy [1]. Similarly, the troubles with building and packaging large and mixed-language codebases are discussed in relation to GNAT Studio [2], and the efforts to port GNAT to NetBSD are described in detail in [3].

For my fellow bookworms out there, two interesting topics can be found in this number: a generous person offered its complete Ada collection [4], providing a exhaustive bibliography worth taking note of, and we got to see a few scanned pages of an old manual in the quest to find a complete Janus/Ada for CP/M online manual [5].

Sincerely,
Alejandro R. Mosteo.

[1] "Challenging a GCC Patch", in Ada-related Resources.

[2] "Building GNAT Studio 2021 from Sources", in Ada Practice.

[3] "Porting Ada to NetBSD", in Ada and Other Languages.

[4] "Ada Books Giveaway", in Ada-related Resources.

[5] "Janus/Ada 1.5 CP/M Manual", in Ada-related Products.

Ada-related Events

30th Anniversary of 1st Ada-Belgium Seminar

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: 30th anniversary of 1st Ada-Belgium Seminar
Date: Sun, 3 Oct 2021 14:47:18 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

Today, October 3rd, 2021, marks the 30th anniversary of the first public event organized by the (then still forming) Ada-Belgium non-profit organization, a half-day Seminar.

A page was created on the Ada-Belgium web-site to present some historic information on that event, retrieved from various archives.

URL: www.cs.kuleuven.be/~dirk/ada-belgium/events/91/911003-abs.html

Enjoy!

Dirk Craeynest, Ada-Belgium President

CfC 26th Ada-Europe Int. Conf. Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 26th Ada-Europe Int. Conf. Reliable Software Technologies
Date: Wed, 20 Oct 2021 19:38:04 -0000
Newsgroups: comp.lang.ada,fr.comp.lang.ada,comp.lang.misc

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

CfP - Ada Developer Room at FOSDEM 2022, Online

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfP - Ada Developer Room at FOSDEM 2022, online
Date: Sun, 5 Dec 2021 10:52:30 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Call for Presentations

11th Ada Developer Room at FOSDEM 2022

Sunday 6 February 2022, Online, Everywhere

www.cs.kuleuven.be/~dirk/ada-belgium/events/22/220206-fosdem.html

Organized in cooperation with Ada-Belgium and Ada-Europe

The Ada FOSDEM community is pleased to announce the 11th edition of the Ada DevRoom This time, however, it will take place online on the 6th of February. This edition of the Ada DevRoom is organized in cooperation with Ada-Belgium [1] and Ada-Europe [2].

General Information about FOSDEM

FOSDEM [3], the Free and Open source Software Developers' European Meeting, is a free and non-commercial two-day weekend event organized early each year in Brussels, Belgium. This year, for obvious reasons, it has been turned into an online event, just like last year. It is highly developer-oriented and brings together 8000+ participants from all over the world. No registration is necessary.

The goal is to provide open source developers and communities a place to meet with other developers and projects, to be informed about the latest developments in the open source world, to attend interesting talks and presentations on various topics by open source project leaders and committers, and to promote the development and the benefits of open source solutions.

Ada Programming Language and Technology

Awareness of safety and security issues in software systems is ever increasing. Multi-core platforms are now abundant. These are some of the reasons that the Ada programming language and technology attracts more and more attention, among others due to Ada's support for programming by contract and for multi-core targets. The latest Ada language definition was updated early 2016. Work on new features is ongoing, such as improved support for fine-grained parallelism, and will result in a new Ada standard scheduled for 2022. Ada-related technology such as SPARK provides a solution for the safety and security aspects stated above.

More and more tools are available, many are open source, including for small and recent platforms. Interest in Ada keeps further increasing, also in the open source community, from which many exciting projects have been started.

Ada Developer Room

FOSDEM is an ideal fit for an Ada Developer Room. On the one hand, it gives the general open source community an opportunity to see what is happening in the Ada community and how Ada can help to produce reliable and efficient open source software. On the other hand, it gives open source Ada projects an opportunity to present themselves, get feedback and ideas, and attract participants to their project and collaboration between projects.

At previous FOSDEM events, the Ada-Belgium non-profit organization organized successful Ada Developer Rooms, offering a full day program in 2006 [4], a two-day program in 2009 [5], and full day programs in 2012-2016 [6-10], and in 2018-2020 [11-13]. An important goal is to present exciting Ada technology and projects, including people outside the traditional Ada community. This edition is no different, and since it will take place online, we hope to attract people from all over the world.

Call for Presentations

We would like to schedule technical presentations, tutorials, demos, live performances, project status reports, discussions, etc in the Ada Developer Room.

Do you have a talk you want to give?

Do you have a project you would like to present?

Would you like to get more people involved with your project?

The Ada organizers call on you to:

- discuss and help organize the details, subscribe to the Ada-FOSDEM mailing list [14];
- for bonus points, be a speaker: the Ada-FOSDEM mailing list is the place to be!
- don't hesitate to propose a topic that you would like to present to the community, we are eager to know what you have in store for us!

We're inviting proposals that are related to Ada software development, and include a technical oriented discussion. You're not limited to slide presentations, of course. Be creative. Propose something fun to share with people so they might feel some of your enthusiasm for Ada!

Speaking slots should be 15 or 30 minutes, plus 5 or 10 minutes resp. for Q&A, if the same schedule as last year is followed. However, this schedule is flexible and can be modified for longer talks. For example, a long technical talk can be transformed into a 45 minutes talk, plus time for Q&A. Depending on interest, we might also have a session with lightning presentations (e.g. 5 minutes each), and/or an informal discussion session.

Note that all talks will be streamed live (audio+video) and should be prerecorded. After the streaming of the talk, a live Q&A session will take place. By submitting a proposal, you agree to being recorded and streamed. You also agree that the contents of your talk will be published under the same license as all FOSDEM content, a Creative Commons (CC-BY) license.

Submission Guidelines

Your proposal must be submitted to the FOSDEM Pentabarf system [15]. If you already had an account from previous years, please, reuse it. If, for whatever reason, you cannot use Pentabarf, you can also submit your proposal by messaging the Ada-FOSDEM mailing list [14]. If needed, feel free to contact us at the Ada-FOSDEM Mailing list or at <irvise (at) irvise.xyz> (without spaces).

Please include:

- your name, affiliation, contact info;
- the title of your talk (be descriptive and creative);
- a short descriptive and attractive abstract;
- potentially pointers to more information;
- a short bio and photo.

See programs of previous Ada DevRooms (URLs below) for presentation examples, as well as for the kind of info we need.

Here is the slightly flexible schedule that we will follow:

- December 26, 2021: end of the submission period. Remember, we only need the information in the list above. You do not have to submit the entire talk by this date. Try to submit your proposal as early as possible. It is better to submit half of the details early than all late, so do not wait for the last minute. If you are a bit late, submit it to Pentabarf and message <irvise (at) irvise.xyz> directly.
- December 31, 2021 - January 2, 2022: announcement of accepted talks.
- January 15, 2022: your talk should be recorded and uploaded to the Pentabarf platform.
- February 6, 2022: Ada-Devroom day!

We look forward to lots of feedback and proposals!

Regards,
The Ada-FOSDEM team

Main organiser: Fernando Oleo Blanco <irvise (at) irvise.xyz>

Second in command: Ludovic Brenta <ludovic (at) ludovic-brenta.org>

----

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium

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

[3] https://fosdem.org

[4] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/06/060226-fosdem.html

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/09/090207-fosdem.html

[6] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/12/120204-fosdem.html

[7] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/13/130203-fosdem.html

[8] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html

[9] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/15/150131-fosdem.html

[10] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/16/160130-fosdem.html

[11] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/18/180203-fosdem.html

[12] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/19/190202-fosdem.html

[13] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/20/200201-fosdem.html

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

[15] https://penta.fosdem.org/submission/FOSDEM22

(V20211205.1)

Ada and Education

"Hello World" as a First Exercise

From: Richard Iswara <haujekchifan@gmail.com>
Subject: Why "Hello World" as a first exercise?
Date: Fri, 30 Jul 2021 13:17:46 +0700
Newsgroups: comp.lang.ada

Why is it most of the courses of introduction to programming or programming language use a "Hello World" kind of program as a demo or first exercise?

Why not do a proper input loop as a showcase or a first exercise? With an input loop procedure you get:

1. How to read and output an input.

2. Show the if-then-else structure.

3. Show the loop structure.

4. Show error messages and how to properly handle it.

5. On Ada in particular you are showing the type system.

6. If it is a subprogram then an input loop shows how to do and call the subprograms.

And last but not least it teaches and reinforces to the student how to think about safety in programming.

So why a useless look at me, ain't I cool "Hello World"?

Sorry I had to vent after an unsatisfying exchange over at arstechnica.

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 30 Jul 2021 02:57:35 -0700

The actual exercise is to (if necessary) get the compiler and tools installed, make the source file, invoke the compiler, and run the executable. Depending on the environment, this can be quite a serious challenge. Going from there to a more complicated program is simple by comparison.

I believe the "hello world" meme started with Brian Kernighan's 1970s-era tutorial for the then-new C language, but I could be wrong about that.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Fri, 30 Jul 2021 02:57:56 -0700

You're right, this Hello World doesn't tell anything about the language, its syntax, semantics, what these have to do with safety. But you find this nonsense, as you say, everywhere.

But how to begin? The opinions vary. They depend on the audience - beginner, experienced...

I say: Give them a simple problem and ask: What kind of (numeric) type do you need to fulfil the needs of the solution.

Others disagree: It's too complicated for a beginner to say "type Meter_Rod is range 0 .. 2_000;", just use Integer for the beginning.

The question is: How to avoid bad habits from other languages from the beginning?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 30 Jul 2021 18:06:21 -0500

> Why is it that most of the courses of introduction to programming or programming language use a "Hello World" kind of program as a demo or first exercise?

Because the problem isn't about programming at all, but rather getting through all of the admistrivia needed to actually run a program. Starting with a canned program of some kind is simply the best plan.

My first actual programming class spent the first two or three sessions on the administrative things: where is the computer center? How do you use a keypunch? (I admittedly am showing my age here; but at least we were the second last semester to use the keypunches.) How to submit a card deck? What magic incantations are needed to get the computer to accept a card desk? Where to find your results afterwards (this being a batch system)? Etc. The actual program was very secondary to all of that (I don't remember what it was, but we had to key it and submit the results — in order to prove that we understood all of the admistrivia).

Obviously, there are differences from then to today, but there still is a lot of admistrivia — both in an academic environment and also at home. (How to use the IDE? How to build a program? How to capture the results? Etc.) So it is very valuable for any student to prove that they understand how to enter and build a trivial program before they turn to actually learning about fundamentals. The flow of any type of course gets interrupted every time someone has problems building a program — the sooner they understand that, the better.

"Hello World" isn't the most interesting program, but it has the advantage of being very short and applicable in most contexts (for instance, it makes sense both in GUI and text environments). And it also shows a primitive way of doing debugging, something that every student will need to know almost from the beginning.

Janus/Ada uses a slightly larger program as an installation test at the end of installation. (At least if you read the installation guide — I wonder how many do? It just sorts a bunch of numbers and displays them to the screen. It's not really a useful example, but it does prove that the Janus/Ada system and the things it depends upon are all installed properly. It doesn't pay to write a program until you are sure of that!

I note that a similar issue happens in a lot of elementary education. I vividly remember that the first word in the first book that we read when learning to read started with an entire page devoted to "Tom" (and a line drawing of a boy). No verb or action or abstraction of any kind. Hardly useful text but valuable in getting the new readers introduced to the idea of text associated with pictures having the same meaning.

The point being that there is a lot of stuff unrelated to the topic at hand that needs to be navigated to learn just about any concept. The sooner that that stuff can be dealt with, the better.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Fri, 30 Jul 2021 21:28:06 -0400

> My first actual programming class spent the first two or three sessions on the administrative things: where is the computer center? How do you use a keypunch? [...]

Sounds like my college... Here are the three 029 keypunches... Here's how to program a drum card to simplify entering code... Here's the minimum JCL to run FORTRAN(-IV) (Sigma CP/V had two FORTRAN compilers — the traditional compiler outputting a relocatable object [ROM] file, to be followed by a linker outputting a load module [commonly called a LMN file]; the OS didn't use file extensions, so our practice was to name the source S:xxx, object O:xxx, executable L:xxx. The other compiler was FLAG — FORTRAN Load And Go — compile/link/execute with one invocation). Turn in the card deck to the operators, here. Come back later to pick up your printed output.

It wasn't until my second year that we were given access to the Hazeltine terminals, along with accounts that had some modicum of disk storage associated with them. The BASIC class got something like 30 "granules" — about 15kB; others got 100-200 "granules".

CP/V somehow combined time-share (we had something like 50 terminals scattered over campus and a few high schools with dial-up lines), Batch Processing, and (not of use to a college installation) supposedly /real-time/ operations.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 31 Jul 2021 20:16:45 -0500

> Sounds like my college... [...]

Ah. We had a very advanced self-service card reader for simple jobs. You put your card deck in, pushed a large button, watched a very impressive swooshing of cards about, and then went and stood around a desk-sized printer with lots of other people waiting for a page with your user name in very large letters to head a printout, rip it off (preferably leaving anyone else's that was attached — didn't always happen), and go read the output to see what you did. The original compile-execute-debug-repeat cycle (more like run-read-punch new cards-repeat cycle).

They had a few Decwriters, but only upper classmen got to use them (and they wasted tons of paper). Real terminals showed up the next year — by the time of the compiler construction class, most of the classes had moved to PDP/11s (way slower), but the compiler construction was still on the mainframe. But almost everything was done on the terminals (Janus/Ada never was on punched cards, thank goodness). We had to buy one of those huge computer tapes to rescue our source code and use another lab's capability to transfer that to floppies in order to move our work to the CP/M computer on which RRS was born. A lot more engineering went into that sort of issue than today (probably a good thing).

From: Richard Iswara <haujekchifan@gmail.com>
Date: Sat, 31 Jul 2021 10:06:17 +0700

> Because the problem isn't about programming at all [...]

Fair points. Obviously now it's a lot different than it was, so why don't textbooks and online instructions, especially those with online IDE, don't evolve their approach?

I still think that students should be trained and challenged to think carefully about the implications of their programs. One of the reservations I have about those online courses or code solutions sites is how many don't consider documentations and coding safely as part of their grades. Skills alone do not suffice in the "real world", communications matter also. How many hours of training after the students graduate will be wasted by their employers to teach them to consider their codes carefully. That is IF (that's the big question) the employers do any kind of training or mentoring. Why isn't that kind of consideration taught and trained until it becomes a habit during the students' education?

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Sat, 31 Jul 2021 19:37:11 -0700

> So why a useless look at me, ain't I cool "Hello World"?

All those things you listed absolutely should be covered — but only *after* the "Hello World" exercise.

The first time someone with no programming experience tries to write, compile, and run a program, *something* will very likely go wrong. Maybe they'll omit a semicolon, or misspell an identifier, or invoke the compiler without a required option. And they're likely to be shown a terse error message that might not direct them to the right way to fix it.

By making the first program something trivial that can reasonably be entered verbatim, you eliminate several sources of errors. If the student double checks that the source file exactly matches what's in the textbook and it doesn't run, it's substantially easier to diagnose the problem.

Once the student gets "Hello, World" working correctly, if the second program uses some of the features you mention and *that* doesn't work, they'll know that the problem is something in the difference between the first and second programs.

You might try a more ambitious first program if you're an experienced programmer trying out a new language, but even then I'll probably try "Hello World" before I try FizzBuzz.

Ada-related Resources

[Delta counts are from Jul 22th to Nov 1st. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Wed, 22 Jul 2021 11:13:21 +0100
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_214 (+53) members	[1]
- Reddit: 7_648 (+544[1]) members	[2]
- Stack Overflow: 2_125 (+38) questions	[3]
- Libera.Chat[2]: 75 (-1) concurrent users	[4]
- Gitter: 91 (+5) people	[5]
- Telegram: 130 (+2) users	[6]
- Twitter: 227 (+152) tweeters	[7]
276 (+202) unique tweets	[7]

[1] Probably caused in part by confusion with the ADA cryptocurrency.

[2] Freenode has been dropped as no data can be obtained anymore.

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

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

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

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

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

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

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

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Wed, 22 Jul 2021 11:13:21 +0100
To: Ada User Journal readership

Rosetta Code:	846 (+19) examples	[1]
	38 (=) developers	[2]
GitHub:	763[1] (=) developers	[3]
Sourceforge:	273 (-2) projects	[4]
Open Hub:	214 (=) projects	[5]
Alire:	195 (+24) crates	[6]
Bitbucket:	88 (-1) repositories	[7]
Codelabs:	53 (+1) repositories	[8]
AdaForge:	8 (=) repositories	[9]

[1] This number is unreliable due to GitHub search limitations.

[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://sourceforge.net/directory/language:ada/

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

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

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

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

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

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Wed, 22 Jul 2021 11:13:21 +0100
To: Ada User Journal readership

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

- TIOBE Index: 31 (-3) 0.42% (-0.06%)	[1]
- PYPL Index: 17 (+1) 0.94% (+0.19%)	[2]
- IEEE Spectrum (general): 31 (+8) Score: 38.8 (+6.0)	[3]
- IEEE Spectrum (embedded): 9 (+3) Score: 38.8 (+6.0)	[3]

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

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

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

AdaControl on Twitter

From: J-P. Rosen <rosen@adalog.fr>
Subject: AdaControl on Twitter
Date: Wed, 1 Sep 2021 17:29:06 +0200
Newsgroups: comp.lang.ada

I have created an account on Twitter for discussing issues related to AdaControl: @AdaControl_prog

(Sorry, @Adacontrol was already taken by a person whose first name is Ada...)

Ada Books Giveaway

From: Michael Feldman <mikefeldman915@gmail.com>
Subject: Giving away a lot of Ada books
Date: Fri, 3 Sep 2021 16:53:48 -0700
Newsgroups: comp.lang.ada

Dear Colleagues,

I hope you are all well and ready for the fall, whatever it might bring in these uncertain times.

My wife Ruth and I are moving from Portland to Berkeley in the near future (yes, our son and his family live there — he is a professor at Cal Berkeley); our new flat will have many advantages, but alas, not nearly enough space for all our books, papers, and media.

I'm writing in the hope of finding new homes for (i.e. giving away) any or all of my large set of books on Ada. I've been collecting these since Ada's beginning in the early 1980s; I think I have all (or very nearly) the Ada books ever published. The list is included below.

Please email me if you're interested in any of these. Ruth has very kindly offered to work with you on the shipping logistics; we ask only that you reimburse us for the shipping costs. Overseas mailing has become such an expensive hassle that I'd prefer the destinations to be in the U.E.

Book dealers don't usually have the understanding or respect for the content of these books, so they have little to no sale value. But they might well have real value to people in the Ada community who know the field. Unfortunately, I fear my options might come down to saying farewell to these treasures on their way to the landfill. I hope not!

Best regards to you and yours; please stay out of COVID's way!

Michael Feldman
Professor Emeritus of Computer Science
The George Washington University
Washington, DC 20052
mfeldman@gwu.edu

==========
Books on the Ada Programming Language (and related topics)
===========

Ada: Berger Tests of Programming Proficiency

AdaTEC Conference 1982

Airiau, R., et al. VHDL: du Langage à la Modernisation

Alsys. Safety Critical Handbook (1994)

Asplund, L. ed. Ada-Europe '98 Proceedings

Audsley, N. Ada Yearbook Millennium Edition

Ausnit, C. et al. Ada in Practice

Baker, L. VHDL Programming

Barnes, J. High Integrity Ada (1997)

Barnes, J. Programming in Ada 95

Barnes, J. Programming in Ada. (2e, 3e, 4e)

Beidler, J. Data Structures and Algorithms (1997)

Ben-Ari, M. Ada for Software Engineers (1998)

Ben-Ari, M. Principles of Concurrent and Distributed Programming (1990)

Ben-Ari, M. Principles of Concurrent Programming (1982)

Benjamin, G. Ada Minimanual (to accompany Appleby, Programming Languages)

Bergé, J-M et al. Ada avec le Sourire

Booch, G. and D. Bryan. Software Engineering with Ada. (3rd edition)

Booch, G. Object-Oriented Analysis & Design, 2nd ed. (1994)

Booch, G. Software Components with Ada.

Bover, D.C.C., K.J. Maciunas, and M.J. Oudshoorn. Ada: A First Course in Programming and Software Engineering.

Bray, G. and D. Pokrass. Understanding Ada.

Breguet, P. and L Zaffalon. Programmation séquentielle avec Ada 95 (in French)

Bryan, D.L., and G.O. Mendal. Exploring Ada, Volumes 1.and 2.

Buhr, R. Practical Visual Techniques in System Design with Applications to Ada.

Burns, A. and A. Wellings. Concurrency in Ada, 2nd ed. (1998)

Burns, A. and A. Wellings. Real-Time Systems and Programming Languages, 2nd ed.

Burns, A. and A. Wellings. Real-Time Systems and Programming Languages, 3rd ed. (2001)

Burns, A. and G. Davies. Concurrent Programming (1993)

Burns, A. Concurrent Programming in Ada.

Burns, Alan and Wellings, Andy. Concurrency in Ada.

Caverly, P. and P. Goldstein. Introduction to Ada.

Cherry, G. Parallel Programming in ANSI Standard Ada

Clark, R. Programming in Ada: a First Course.

Cohen, N. Ada as a Second Language.

Computer Language, March 1989 (compilers)

Cooling, J.E. et al. Introduction to Ada

Crawford, B.S. Ada Essentials

Culwin, F. Ada: a Developmental Approach. (2nd ed)

Dale, N., D. Weems, and J. McCormick. Programming and Problem Solving with Ada. D. C. Heath, 1994.

Dale, N., S. Lilly, and J. McCormick. Ada plus Data Structures.

Defense Electronics, March 1984 - first Ada compilers

DeLillo, N. J. A First Course in Computer Science with Ada.

Denev, N. Programming (in Bulgarian)

DISA Symposium on Ada Success in MIS (1992)

Dorchak, S. and P. Rice. Writing Readable Ada

Downes, V. and S. Goldsack. Programming Embedded Systems with Ada.

Embedded Systems Programming, Nov. 1995 (Ada 05 issue)

English, J. Ada 95: the Craft of Object-Oriented Programming (1997)

Feldman, M. Concepts of Concurrent Programming (1990)

Feldman, M. Language and System Support for Concurrent Programming (1990)

Feldman, M.B. Data Structures with Ada.

Feldman, M.B. Software Construction and Data Structures with Ada 95 (1996)

Feldman, M.B., and E.B. Koffman. Ada: Problem Solving and Program Design.

Feldman/Koffman Ada 95 (1st printing, 3rd printing)

Freedman, R. Programming Concepts with the Ada Language.

Gabrini, P. Introduction au Génie Logiciel et à la Programmation avec Ada (in French)

Gauthier, M. Ada: a Professional Course.

Gauthier, M. Ada: Un Apprentissage (in French).

Gehani, N. Ada: an Advanced Introduction (2nd edition).

Gehani, N. Ada: Concurrent Programming (2nd edition).

Gehani, N. and A. McGettrick. Concurrent Programming (1988)

Gehani, N. and W.D. Roome. The Concurrent C Programming Language (1989)

Gilpin, G. Ada: a Guided Tour and Tutorial.

Glynn, G. Ada Yearbook 1998

Gonzalez, D. Ada Programmer's Handbook

Habermann, A. and D. Perry. Ada for Experienced Programmers.

Hardy, N. Ada Yearbook 1996

Hibbard, P. et al. Studies in Ada style

Hillam, Bruce. Introduction to Abstract Data Types Using Ada.

HOPL-II Forum on the History of Computing preprints (1993)

IBM Software Engineering Exchange, Oct. 1980 (Ada edition)

IBM Systems Journal 1991, 25th anniversary of APL

Jones, D. Ada in Action

Jonston, S. Ada 95 for C and C++ Programmers (1997)

K.U. Leuven Dept. of CS Report 91-92

Krell, B. Developing with Ada

Lamprecht, G. Introduction to Simula-67 (1983)

Lindsey, E.R. The Encyclopedic Dictionary of Ada terms

Lomuto, N. Problem-Solving Methods with Examples in Ada.

Lopes, A.V. Ada 95 (in Portuguese)

Lundqvist, K. Distributed Computing and Safety-Critical System in Ada (diss.)

Mayoh, B. Problem Solving with Ada.

Miller, N.E. and C.G. Petersen. File Structures with Ada.

Motet, G. et al. Design of Dependable Ada Software

Musser, D. and A. Stepanov. The Ada Generic Library

Naiditch, D.H. Rendezvous with Ada 95

Naiditch, D.J. Rendezvous with Ada

National Academy of Sciences, Ada and Beyond (1997)

Naur, P. and B. Randell. NATO Conference on Software Engrg (1969)

Nielsen, K. Object-Oriented Design with Ada

Nissen, J. and P. Wallis. Portability and Style in Ada.

Nyberg, K. (editor) The Annotated Ada Reference Manual. (2nd edition)

Olsen, E. and S. Whitehill. Ada for Programmers.

Perminov, O. Programming in Ada (in Russian)

Price, D. Introduction to Ada.

Pyle, I. The Ada Programming Language.

Rosen, J-P and Kruchten, P. Doctoral Theses on Ada/Ed

Rosen, J-P. Méthodes de Génie Logiciel avec Ada 95 (in French)

Saib, S. Ada: an Introduction.

Sanden, B. Software Systems Construction with Examples in Ada

Savitch, W.J. and C.G. Petersen. Ada: an Introduction to the Art and Science of Programming.

Saxon, J.A., and R.E. Fritz. Beginning Programming with Ada

Schneider, G.M., and S.C. Bruell. Concepts in Data Structures and Software Development (with Ada Supplement by P. Texel).

Shumate, K. Understanding Ada. (2nd edition)

Shumate, K. Understanding Concurrency with Ada.

SIGCSE Bulletin June 1991

SIGPLAN Notices June 1979 (A) Ada Proposed Rationale

SIGPLAN Notices June 1979 (A) Ada Proposed Reference Manual

Skansholm, J. Ada 95 from the Beginning (3rd ed)

Skansholm, J. Ada from the Beginning. (2nd ed.)

Smith, M. Object-Oriented Software in Ada 95 (1996)

Software Productivity Consortium, Ada 95 Quality and Style (1995)

SpAda

Stein, D. Ada: a Life and a Legacy (1985)

Stratford-Collins, M.J. Ada: a Programmer's Conversion Course (in Chinese)

Strohmeier, A. Ada Software Components (1992)

Stubbs, D., and N. Webre. Data Structures with Abstract Data Types and Ada

Tedd, M. et al. Ada for Multi-microprocessors

Texel, P. Introductory Ada. (1986)

Thiess, H. Minimal Ada

Toole, A. Ada, the Enchantress of Numbers (2 copies) (1998)

Tremblay, J-P et al. Programming in Ada (1990)

US Navy Ada Implementation Guide

Vasilescu, E. Ada Programming with Applications.

Volper, D., and M. Katz. Introduction to Programming Using Ada.

Wallach, Y. Parallel Processing and Ada

Watt, D.A., B.A. Wichmann, and W. Findlay. Ada Language and Methodology.

Wegner, P. Programming with Ada.

Weiss, M.A. Data Structures and Algorithms in Ada.

Wheeler, D. Ada 95, the Lovelace Tutorial (1997)

Young, S. An Introduction to Ada.

Zaffalon, L. and P. Breguet. Programmation Concurrente et temps réel avec Ada 95 (in French)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 3 Sep 2021 22:28:50 -0500

Good to hear from you again.

It must have taken a long time to simply list all of these; thanks for not throwing these away.

I have quite a collection of Ada books here, but you have me beat by a long way.

I'd be interested in Norm Cohen's Ada as a Second Language; at one point, we gave our copy to a newly hired person to study and they never brought it back. (Most of the early textbooks we had multiple copies of; we used to sell Ada books to our customers and still have a few leftovers that were not sold.) E-mail me if you haven't given it to someone else.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 05 Sep 2021 11:15:54 -0700

The University of California at Berkeley library (https://www.lib.berkeley.edu/

there's a link to a "contact us" page) might want some of these. If not, take them to your local library; they might just trash them, but they might also end up on a shelf somewhere ...

Challenging a GCC Patch

From: J-P. Rosen <rosen@adalog.fr>
Subject: How to challenge a GCC patch?
Date: Mon, 27 Sep 2021 12:06:56 +0200
Newsgroups: comp.lang.ada

AdaCore has introduced a patch in FSF GCC to remove ASIS support.

AdaCore is free to do what they want with their own version of GCC. However, removing a useful feature from the FSF version with the goal to promote their own, in-house tool is clearly against the spirit of free software.

Does anybody know the procedures set by the FSF to challenge a patch?

From: Stéphane Rivière <stef@genesix.org>
Date: Mon, 27 Sep 2021 13:23:38 +0200

> AdaCore has introduced a patch in FSF GCC to remove ASIS support.

This is all the more surprising since it seems to me that ASIS is still in GNAT Pro. What a lack of fairness.

> Does anybody know the procedures set by the FSF to challenge a patch?

Unfortunately no

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 27 Sep 2021 16:18:30 +0200

> This is all the more surprising since it seems to me that ASIS is still in GNAT Pro. What a lack of fairness.

ASIS is no more in the mainstream gcc, it's in a special version, forked from the main branch, called asis-gcc.

See the instructions on running AdaControl for details: https://www.adacontrol.fr

From: Simon Wright <simon@pushface.org>
Date: Mon, 27 Sep 2021 13:48:30 +0100

> AdaCore has introduced a patch in FSF GCC to remove ASIS support. [...]

It's not just the patch(es), it's any subsequent changes to affected parts of the compiler.

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 27 Sep 2021 16:20:05 +0200

> It's not just the patch(es), it's any subsequent changes to affected parts of the compiler.

Right, if they want to contribute further patches, they'll have to keep it ASIS compatible. That's not a reason to divert gcc to support their own private interests.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Mon, 27 Sep 2021 23:55:38 -0700

I must admit I fail to see your point in this thread: as far as I know, ASIS has never worked for recent versions of the language (standard was never updated), and AdaCore doesn't not evolve it anymore. Yes, that unfortunately means that tools like AdaControl will stop working at some point (you can certainly distribute prebuilt binaries for a while, but for anyone using new language constructs, what happens?). This being open-source software, you could adopt the maintenance of ASIS yourself (or ask other people in the Ada community to help with that). But this is of course a significant endeavor (then again, if you are not ready to do that yourself, why would you expect a commercial company like AdaCore to do it on your behalf?)

ASIS has not disappeared. It is still (and forever) in the history of the gcc tree. It is just not available on the main branch anymore because there are no more maintainers for it. Just like a lot of obsolete platforms no longer supported by gcc itself, or by the Linux kernel for instance. This is the way open-source software lives and dies.

Going back to a more technical discussion, would you highlight why a library like libadalang is not appropriate for AdaControl. I have developed a few code-generation tools based on it. To me, the main issue is the bad documentation, which leaves a lot of trial-and-error to find which nodes are relevant when. Besides that, it seems to be fine with any code I have sent its way. Maybe, rather than trying to maintain your own ASIS patches, it would be nice to develop an ASIS API that uses libadalang underneath (I do not know much about ASIS to be honest, so this might be a stupid suggestion).

From: Arnaud Charlet <charlet@adacore.com>
Date: Tue, 28 Sep 2021 00:38:32 -0700

> AdaCore has introduced a patch in FSF GCC to remove ASIS support.

We have removed ASIS support first in our own trunk of GNAT, and then 6 months later we have removed it from the GCC FSF trunk, so talking about lack of fairness is, well, unfair.

Why? Because ASIS is no longer maintained as an internal standard and hasn't evolved beyond Ada 95 because there was not enough support in the community and among vendors, so we've ended up maintaining it on our own for many years, which lately has become too large a burden. In addition, maintaining ASIS tree generation in GNAT has been also a challenge and a resource drain because each time we make a change in the GNAT front-end, this may break ASIS and we may have to make difficult investigation and changes and sometimes almost impossible changes because there are conflicts between the need of a code generator (GNAT for GCC or LLVM) and the need of an Ada analysis library (ASIS).

So we've decided to address this burden by moving tree generation for ASIS in a separate branch, so that this maintenance burden on GCC trunk would disappear.

This has been done both in AdaCore's tree where ASIS now resides on a separate branch, and in GCC FSF where the tree generation is available in GCC 10.x and works well here, and is available for the community to contribute and maintain for as long as needed.

From: Stéphane Rivière <stef@genesix.org>
Date: Wed, 29 Sep 2021 18:26:04 +0200

> We have removed ASIS support first in our own trunk of GNAT, and then 6 months later we have removed it from the GCC FSF trunk, so talking about lack of fairness is, well, unfair.

I deeply endorse your maintenance and code evolution concerns.

The lack of 'fairness' (my apologies if you find that word a bit strong) is that GNAT Pro users are suddenly the only ones who can use ASIS, while a unique tool like Adacontrol (for code control quality) has always been available equally to the Free and Pro communities...

> Why? Because ASIS is no longer maintained as an internal standard and hasn't evolved beyond Ada 95 because there was not enough support in the community and among

Thanks Arno for these explanations...

We all know about Adacore's commitment to the Free Software community. The latest versions of GNATStudio, which has never been so reliable and user-friendly, are just one example among others.

However, the initial problem persists and cannot be solved quickly.

Should Adacontrol users find a relationship using GNAT-Pro to release the tools needed to continue using Adacontrol?

Maybe AdaCore could reconsider its decision to keep ASIS for the Pro community only and release it again, at least temporarily, to give the Libre community some time to find a sustainable solution?

This could be an intermediate solution, pending a possible port of Adacontrol to libadalang (or any other satisfying way).

Perhaps AdaCore could help the community and Jean-Pierre in this process? (targeted help, improved documentation, etc.)?

Thanks again for participating in this thread. It is very interesting to talk with a representative of the most essential Ada contributor to Libre software.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Wed, 29 Sep 2021 12:04:07 -0700

> The lack of 'fairness' (my apologies if you find that word a bit strong) is that GNAT Pro users are suddenly the only ones who can use ASIS [...]

I might have misunderstood Arno's point, but my understanding is that AdaCore no longer makes any patch for ASIS. So whatever pro customers have access to (and ASIS was always a paying addon), the community also has access to by downloading the latest available sources.

The GNAT Pro compiler apparently is losing the capability to generate the tree information, just like the free version of the compiler. If you want to use ASIS, my understanding is that you would have to do a separate "compilation" pass using the compiler from the dedicated branch just for the purpose of generating the tree files (and you can discard all the object files it perhaps generates at the same time). Then you can run ASIS tools.

This is for sure a pain for AdaControl maintainers and users, no one disputes that. On the other hand, if tree generation was indeed getting in the way of compiler improvements that benefit everyone, I, for one, am happy to see the change.

> Perhaps AdaCore could help the community and Jean-Pierre in this process? (targeted help, improved documentation, etc.)?

I suggested in an early message that perhaps the community could build an ASIS API on top of libadalang, if there is a need for that.

I also suggested that libadalang documentation should be improved, I definitely agree with that one!

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 30 Sep 2021 00:29:10 +0100

> I suggested in an early message that perhaps the community could build an ASIS API on top of libadalang, if there is a need for that.

Freely available ISO ASIS spec would help here.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 08:23:17 +0200

> Freely available iso asis spec would help here.

Actually, it is. Apart from ISO verbiage, all the interesting parts of the ASIS standard are put as comments in the corresponding ASIS packages.

Moreover, AdaCore kept this good habit for all the newly introduced features that support up to Ada 2012, which would make retrofitting them into an updated ASIS standard quite easy.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 07:57:26 +0200

> Why? Because ASIS is no longer maintained as an internal standard and hasn't evolved beyond Ada 95 [...]

The ASIS standard has not been updated, but AdaCore did a great job of evolving its ASIS implementation to support all new features up to Ada2012. It would be easy to add these improvements to a revised ASIS standard, and a New Work Item will be proposed to ISO to that effect.

Anyway, this issue of ASIS not being an up-to-date standard is a red herring, since LibAdalang is NOT a standard, and presumably never will.

> so we've ended up maintaining it on our own for many years, which lately has become too large a burden.

This is plain wrong. You don't maintain ASIS "on your own", there are customers who pay a support contract for ASIS.

> [...] So we've decided to address this burden by moving tree generation for ASIS in a separate branch, so that this maintenance burden on GCC trunk would disappear.

We are talking about FSF-GNAT here. AFAIK, asis-gcc has not been pushed to FSF-GNAT.

> This has been done both in AdaCore's tree where ASIS now resides on a separate branch, and in GCC FSF where the tree generation is available in GCC 10.x and works well here, and is available for the community to contribute and maintain for as long as needed.

But this means that users of ASIS will be stuck to GCC 10.x, or will have to handle two versions of gcc at the same time, which is an endless source of burden. Why don't you make asis-gcc available to the community? It doesn't require any extra cost, since it is available to paying customers!

Anyway, my question was about how to challenge a patch. I estimate that this patch is unfortunate, you argue that it is necessary. Let the GCC governance decide; AdaCore doesn't rule GCC.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 08:19:30 +0200

> I might have misunderstood Arno's point, but my understanding is that AdaCore no longer makes any patch for ASIS.

No, ASIS is still maintained (although as LTM) for paying customers.

> So whatever pro customers have access to (and ASIS was always a paying addon), the community also has access to by downloading the latest available sources.

No, asis-gcc is not distributed by AdaCore.

> If you want to use ASIS, [...] you would have to do a separate "compilation" pass using the compiler from the dedicated branch just for the purpose of generating the tree files [...] Then you can run ASIS tools.

Not really. Compile-on-the-fly is still working with asis-gcc (AdaControl is working like that).

> This is for sure a pain for AdaControl maintainers and users, no one disputes that. On the other hand, if tree generation was indeed getting in the way of compiler improvements that benefit everyone, I, for one, am happy to see the change.

I'm afraid this is a red herring. I rather think that AdaCore has a hard time convincing people of moving from the well defined, carefully designed ASIS to the terrible mess of LibAdalang.

To anybody interested in that issue: don't take my word for it. Please read the specification of any ASIS module, and compare it to the libadalang.analysis package.

Personally, I will never trust an interface that documents that I should expect a character literal on the LHS of an assignment statement!

Another example: it's only very recently (not sure if it is already in GitHub) that LibAdalang considered the case of a variable declaration with multiple names. How do you explain such an omission after 5 years of development?

>> Perhaps AdaCore could help the community and Jean-Pierre in this process? (targeted help, improved documentation, etc.)?

I have had a tool partner's agreement with AdaCore, and until recently they have been very helpful. But the whole design of LibAdalang is not appropriate for deep static analysis, and it is an error to believe that it could replace ASIS. OTOH, it has plenty of useful features for other use cases not covered by ASIS, like handling of incomplete/incorrect code, no question about that.

> I suggested in an early message that perhaps the community could build an ASIS API on top of libadalang, if there is a need for that.

In the beginning of LibAdalang, AdaCore suggested doing that, but they abandoned it.

> I also suggested that libadalang documentation should be improved, I definitely agree with that one!

Unfortunately, the whole design (and especially the typing system) of Libadalang makes it much more difficult to use than ASIS.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 08:44:49 +0200

> I must admit I fail to see your point in this thread: as far as I know, ASIS has never worked for recent versions of the language (standard was never updated), and AdaCore doesn't not evolve it anymore.

Your information is not up-to-date. AdaCore has evolved its ASIS implementation to fully support up to Ada 2012, and there will be a proposal to renew the ASIS standard at ISO.

Claiming that ASIS is obsolete and has not evolved since 95 is pure FUD propagated by AdaCore. Anybody can download AdaCore's latest implementation and check that Ada 2012 is fully supported.

> Yes, that unfortunately means that tools like AdaControl will stop working at some point [...]

AdaControl fully supports Ada 2012. Many new features of Ada 202x use aspects, which are fully supported. The main syntactic addition is the "parallel" constructs, but few people will need it, and AdaCore said once that they would not support it.

> [...] why would you expect a commercial company like AdaCore to do it on your behalf?

Because that commercial company has customers who pay for that.

[...]

> [...] would you highlight why a library like libadalang is not appropriate for AdaControl. [...]

1) the typing system. Yes, the typing system of ASIS is surprising at first sight, but extremely convenient to use. I suspect that the designers of LibAdalang never studied the rationale behind ASIS choices when they decided to make that huge hierarchy of tagged types that brings no more static checks (you still need checks at run-time that elements are appropriate for their usage), but makes a lot of things more difficult. As an example, there are plenty of simple loops in AdaControl that would need to be changed to recursive calls of special functions (one for each loop).

2) Missing features. A casual look-up showed a number of queries that I could not find. I reported to AdaCore, the response was: "yes, that's a good idea, we'll add that later".

3) Unfriendly interface. It's not only lack of documentation, the "P_" and "F_" convention makes everything harder to read, and is of no benefit to the user. Moreover, it is a matter of implementation that surfaces to the specification - very bad. Where ASIS strictly follows the terms and structure of the ARM, LibAdalang uses abbreviated names that do not even correspond to the usual Ada vocabulary. And this cannot be fixed without a major, incompatible, rework.

From: Arnaud Charlet <charlet@adacore.com>
Date: Thu, 30 Sep 2021 00:29:26 -0700

> We are talking about FSF-GNAT here. AFAIK, asis-gcc has not been pushed to FSF-GNAT.

What you call "asis-gcc" is a Pro version. We've never pushed any Pro version to FSF-GNAT, and there has never been any guarantee of correspondence between GNAT Pro and FSF-GNAT, so what you are demanding today for ASIS is unreasonable and unnecessary.

So assuming you are asking instead for some FSF version "close to asis-gcc", this version is available in the GCC 10.x branch, and similarly to asis-gcc which is on a long term, low changes branch at AdaCore, GCC 10.x is in the same state today. If you want an executable called "asis-gcc" then make a symbolic link from gcc (10.x) to asis-gcc and you have it.

> But this means that users of ASIS will be stuck to GCC 10.x, or will have to handle two versions of gcc at the same time, which is an endless source of burden.

The same is true for Pro users, no difference here: Pro users need to use GNAT x to compile, and ASIS-GCC y to generate trees. So what you are complaining about isn't different between Pro and community users, and making asis-gcc Pro available won't change that.

So to recap: you are asking for a Community version of "asis-gcc Pro": this version is available, it's GCC 10.x (10.3 being the latest available to date). And yes, it's a different version to generate trees than to compile Ada: the same is true for Pro users and they do not have specific issues with that.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 09:52:36 +0200

> So to recap: you are asking for a Community version of "asis-gcc Pro": this version is available, it's GCC 10.x [...]

But it's not available from AdaCore's community page. For most users, downloading and building from an FSF site is way too complicated. Call it asis-gcc or not, what is needed is a simple way to install ASIS support.

(Making a tree generator separate from the compiler is for me another error, although I can live with it. One of the main benefits of ASIS is that the ASIS program has the same view of the code as the compiler - but that's a separate issue).

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 30 Sep 2021 08:53:06 +0100

> Moreover, AdaCore kept this good habit for all the newly introduced features that support up to Ada 2012, which would make retrofitting them into an updated ASIS standard quite easy.

Are they GPL'd and where are they?

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 10:13:58 +0200

Yes. Here is a copy of the copyright notice of every ASIS module:

```
-- This specification is adapted from the
-- Ada Semantic Interface Specification
-- Standard (ISO/IEC 15291) for use with
-- GNAT. In accordance with the
-- copyright of that document, you can
-- freely copy and modify this
-- specification, provided that if you
-- redistribute a modified version, any
-- changes that you have made are clearly
-- indicated.

-- This specification also contains
-- suggestions and discussion items
-- related to revising the ASIS Standard
-- according to the changes proposed for
-- the new revision of the Ada standard.
-- The copyright notice above, and the
-- license provisions that follow apply
-- solely to these suggestions and
-- discussion itemas that are separated by
-- the corresponding comment sentinels
```

[More standard GPL 2 text omitted. —arm]

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 10:16:22 +0200

> Are they GPL'd and where are they?

You can find them in the specifications of the various packages, with sentinels (as indicated in my previous message). Another excerpt:

```
-- Suggestions related to changing this
-- specification to accept new Ada
-- features as defined in incoming
-- revision of the Ada Standard
-- (ISO 8652) are marked by following
comment sentinels:

-- --|A2005 start

-- ... the suggestion goes here ...
```

-- --|A2005 end

-- and the discussion items are marked by
-- the comment sentinels of the form:

-- --|D2005 start

-- ... the discussion item goes here ...

-- --|D2005 end

(and the same goes for 2012).

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 30 Sep 2021 09:26:12 +0100

> Yes. Here is a copy of the copyright notice of every ASIS module:

And that's an issue, why not release them PD or BSD? I've seen the ASIS specs before and I'm certain they are not GPL'd, just like the packages in the Ada RM.

From: Arnaud Charlet <charlet@adacore.com>
Date: Thu, 30 Sep 2021 01:21:35 -0700

> But it's not available from AdaCore's community page. For most users, downloading and building from an FSF site is way too complicated. Call it asis-gcc or not, what is needed is a simple way to install ASIS support.

We have decided in any case to stop creating and distributing GNAT Community binaries, since this was causing too much confusion and misunderstanding wrt the license, doing in the end more harm than good to the community, which we care very much about.

So in the future, GNAT will be available directly and only from the FSF versions, and Alire will make that easy.

Alire (https://alire.ada.dev/) already provides GCC 10.3 today, see e.g. "alr toolchain --select"

> (Making a tree generator separate from the compiler is for me another error, although I can live with it. One of the main benefits of ASIS is that the ASIS program has the same view of the code as the compiler - but that's a separate issue).

Right, and has never been the case for cross compilers where you already needed a native GNAT to build your ASIS application, and a cross GNAT to generate trees.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Thu, 30 Sep 2021 01:28:16 -0700

> But it's not available from AdaCore's community page. For most users, downloading and building from an FSF site is way too complicated.

There are plenty of GNAT FSF 10 builds available:

- Linux distribs (Ubuntu/Debian, Arch, Fedora, and probably others that I don't know about)
- msys2 for Windows
- Simon Wright's builds for macOS
- Alire for Linux, Windows and macOS

Availability of GNAT FSF 10 is not an issue.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 12:54:33 +0200

> And that's an issue, why not release them PD or BSD? I've seen the asis specs before and I'm certain they are not GPL'd, just like the packages in the Ada RM.

If you are talking about the official ASIS specs ("like the packages in the Ada RM"), they are part of an ISO standard, and as such under ISO copyright. However, in the case of APIs, ISO allows their use by any implementation (otherwise, they would be useless). This has of course nothing in common with the GPL or any other open license.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 12:56:52 +0200

> I wanted to know where they are. I once found the entire directory of ASIS specs from the iso doc, I think I have them somewhere still.

> Where are the updated ones for post 95? There should be an archive or directory with them with no restrictive licensing comments.

Just download ASIS for GNAT CE 2019.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 30 Sep 2021 13:27:11 +0100

> If you are talking about the official ASIS specs ("like the packages in the Ada RM"), they are part of an ISO standard, and as such under ISO copyright. However, in the case of APIs, ISO allows their use by any implementation (otherwise, they would be useless).

Exactly, same as the ARM packages.

> This has of course nothing in common with the GPL or any other open license.

But the issue is, if the specs for the extended ASIS have only been released under GPL, they are useless to any non-gpl language implementations as their use infects that implementation causing further issues.

This GPL issue is the reason why I've looked at, in the past, creating my own compiler, and now just wanting to develop my own language that I can use anywhere.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 30 Sep 2021 13:27:37 +0100

>> Where are the updated ones for post 95? There should be an archive or directory with them with no restrictive licensing comments.

> Just download ASIS for GNAT CE 2019.

No. See my other message.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 17:28:39 +0200

If you want a separate, available document, I don't think there is. If it is just out of curiosity, use ASIS for GNAT.

There is certainly work to do to get an updated standard!

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 30 Sep 2021 17:25:25 +0200

> But the issue is, if the specs for the extended ASIS have only been released under GPL, they are useless to any non-GPL language implementations as their use infects that implementation causing further issues.

Right, currently AdaCore is the owner of these specifications. A standardization effort would need a transfer of copyright, I hope that AdaCore wouldn't object.

BTW, talking of copyright: LibAdalang has no header comment telling the copyright status, therefore it is by default proprietary AdaCore!

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 30 Sep 2021 19:18:05 -0500

> I'm afraid this is a red herring. I think rather that AdaCore has a hard time convincing people of moving from the well defined, carefully designed ASIS to the terrible mess of LibAdalang.

The ASIS design and definition is a mess (at least from the perspective of explaining what is expected). We tried to clean it up in the previous ASIS standardization update, but that was a lot of work and we probably didn't match implementations very well.

The entire model of ASIS doesn't make much sense for static analysis purposes, it's way too focused on syntax rather than semantics. And it doesn't work well for syntax analysis because it requires a compilable program. So it really has a very narrow use case (if any).

Your tool mainly proves that one can use anything with heroic enough efforts. But the effort that your tools goes through to determine basic semantics like whether a type is tagged demonstrates it's hardly a practical way to build a tool. As far as I know, you're the only one that ever managed to do anything beyond proof-of-concepts with ASIS. I can certainly see why AdaCore might not want to support something solely for one usage.

I can easily believe that Libadalang is even more poorly defined than ASIS (most vendor-generated things are, regardless of the vendor involved). I would guess that the only way to build a tool like yours is to do your own analysis (certainly, that is how I'd approach it). A true Ada Semantic Interface would be a good thing, but ASIS isn't it.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 30 Sep 2021 19:30:03 -0500

> This specification is adapted from the Ada Semantic Interface Specification Standard (ISO/IEC 15291) for use with GNAT. [...]

Umm, someone is confusing the original ASIS drafts with the ISO Standard (which has an ISO copyright with no exceptions). I would definitely not reference the ISO Standard in anything you are freely giving away -- there are copyright trolls out there that could easily decide to get your material banned from the Internet.

For Ada, we are very carefully keeping the Ada Reference Manual as a separate document from the ISO Standard, so that the Ada RM has the permissive copyright while the ISO Standard for Ada definitely does not. These are not the same thing!

That care was not taken for the ASIS Standard; I know of no public version that was maintained. As such, my opinion is that ISO owns the copyright, and any extensive use (like using all of the specs) would require a license from ISO. This is by far the best reason for abandoning ASIS - I don't believe that you can implement it without getting a license from ISO (since the bulk of the ASIS Standard is Ada specifications, you are using too much to fall under fair use). This is one reason that I would never consider implementing ASIS in Janus/Ada.

> Moreover, AdaCore kept this good habit for all the newly introduced features that support up to Ada 2012, which would make retrofitting them into an updated ASIS standard quite easy.

It's only easy if you think that giving AdaCore's work to ISO under the exclusive copyright that they (ISO) will insist on is something that is legally and ethically appropriate.

You need to come to grips with the reality that ASIS is dead. It's legally dangerous to implement it, it isn't a good match for either syntax or semantic analysis (doing neither very well), and it is a poor match for modern compilers (hardly anyone builds trees much like the ASIS ones, unless you are trying to implement ASIS).

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 1 Oct 2021 11:24:15 +0200

> The ASIS design and definition is a mess (at least from the perspective of explaining what is expected). We tried to clean it up in the previous ASIS standardization update [...]

That was mainly an attempt to introduce more static and tagged typing, and it failed due to the complexity involved (and that AdaCore said they would never implement it). LibAdalang made the same error, and got the same unnecessary complexity.

> The entire model of ASIS doesn't make much sense for static analysis [...]

It is an exact image of the program, from which you can derive all the information you need. Some higher level queries are needed, but they can be provided as secondary queries or added to the standard.

> And it doesn't work well for syntax analysis because it requires a compilable program. So it really has a very narrow use case (if any).

On the contrary. There is no semantic you can analyze in a non-compilable program. And since it analyzes the output of a validated compiler, you can trust it better than any custom analyzer without known pedigree.

> the effort that your tools go through to determine [...] whether a type is tagged demonstrates it's hardly a practical way to build a tool.

I'm afraid you are confused here. It is very easy to check whether a type is tagged. You may be confusing this with checking whether a type is limited or not: yes, an extra query would be useful for this case. No big deal.

> As far as I know, you're the only one that ever managed to do anything beyond proof-of-concepts with ASIS.

For years, AdaCore tools (gnatelim, gnatstub) used ASIS, not counting Gnatcheck that has not yet been able to migrate to LibAadalang. The interface generator of AWS is also based on ASIS. Out of the top of my mind, I think certain document generators as well as some real-time properties analyzers also use ASIS.

[...]

True, a first approach or a casual reading of the interface is not very friendly. But the more you use it, the more you realize that it is very consistently defined, and allows you to do whatever you need.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 1 Oct 2021 11:41:05 +0200

> Umm, someone is confusing the original ASIS drafts with the ISO Standard (which has an ISO copyright with no exceptions). I would definitely not reference the ISO Standard in anything you are freely giving away

Strangely enough, my copy of ISO 15291 has no copyright statement at all; might be a "last draft" version.

However, the headers of every ASIS-for-Gnat package state: "This specification is adapted from the Ada Semantic Interface Specification Standard (ISO/IEC 15291) for use with GNAT. In accordance with the copyright of that document, you can freely copy and modify this specification, provided that if you redistribute a modified version, any changes that you have made are clearly indicated."

(and since that statement dates back to Robert Dewar's times, I'm pretty certain it is reliable).

My memory is that all "interesting" part of the standard was deliberately put as comments in the specification, precisely to circumvent the ISO copyright, and allow the use of ASIS without paying an outrageous price to ISO.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 1 Oct 2021 11:56:48 +0200

> We have decided in any case to stop creating and distributing GNAT Community binaries [...]

And what will happen to other versions of GNAT that were useful for promoting Ada, like JGnat and Lego-mindstorm? (I know you froze these some years ago, but it was very useful to be able to mention them).

And what will happen for fixes to asis-gcc? Will they be propagated to GCC 10.3? Even after you move to GCC 11.x?

> Alire (https://alire.ada.dev/) already provides GCC 10.3 today, see e.g. "alr toolchain --select"

And does it provide a matching version of ASIS?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 2 Oct 2021 04:14:59 -0500

> However, the headers of every ASIS-for-Gnat package state:

> "This specification is adapted from the Ada Semantic Interface Specification Standard (ISO/IEC 15291) for use with GNAT.

I'm certain that is something that predates the ISO version of ASIS. There's no such permission in the ISO document that I was sent as editor during our last (aborted) revision attempt. Robert probably was using the pre-ISO version as the source, all

> My memory is that all "interesting" part of the standard was deliberately put as comments in the specification, precisely to circumvent the ISO copyright, and allow the use of ASIS without paying an outrageous price to ISO.

I don't see how using comments helps anything. The Oracle case makes it pretty clear an API itself can be covered by a copyright, and surely the comments are covered by the copyright. And the ISO version has no copyright statement other than the usual "All rights reserved".

Disclaimer: I am not a lawyer and cannot say anything for certain in these matters.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 2 Oct 2021 04:34:30 -0500

>> The entire model of ASIS [...]

> It is an exact image of the program, from which you can derive all the information you need.

That's exactly the problem. You start with the source code, which is way too low a level for any useful analysis. At most, you want a simple connection to the source in the semantic information, not trying to preserve every punctuation mark and comment. (Janus/Ada discards all of that stuff as soon as parsing succeeds.) If you need to refer to the original source, say for error handling purposes, then do that, but don't waste vast amounts of space and time trying to keep loads of irrelevant material.

[...]

No sane compiler (validated or not) keeps all of the irrelevant syntactic detail required by ASIS. It ends up getting reconstructed solely for the use of ASIS, and how a rarely used interface is somehow more reliable escapes me.

A true semantic interface on the lines of the one proposed for ASIS would make good sense (design of types), but the vast majority of the existing ASIS belongs in a rubbish bin. Good riddance.

> But you didn't use it.

I don't use it because implementing it would require adding loads of useless cruft to our Ada compiler. And even then, it doesn't make much sense based on our compilation model and our generic unit model. Supporting it would be like building a whole new Ada compiler. Ergo, it is a lie, it claims to be an "interface to a compiler", but it requires many things that most compilers would not waste time on. (I think it is a fairly close representation of the internals of early Rational compilers, which is probably why they were so slow and memory hogs. ;-) So what is it really? Just a very complex way to do stuff that you can easily do with a parser. Not worth anyone's time, IMHO.

I've assumed most people used it because it was there and because some people had spent a lot of time trying to define it as some sort of Standard. Just because people put a lot of work into something doesn't mean that it is a useful thing.

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 4 Oct 2021 14:26:25 +0200

[...]

> [...] the vast majority of the existing ASIS belongs in a rubbish bin. Good riddance.

Says someone who didn't use or implement ASIS. BTW, I understand that ASIS would be difficult to implement in Janus Ada, especially when it comes to generic expansion. But it's not a reason to deprive others from it...

> [...] (I think it is a fairly close representation of the internals of early Rational compilers

True, the design was based on ideas from Diana. But it was designed with inputs from various compilers.

> Just because people put a lot of work into something doesn't mean that it is a useful thing.

It allowed me to build a very sophisticated tool, valued at 1.24M$ (see https://www.adacontrol.fr), and used by very serious customers. Seems enough to qualify it "useful".

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 4 Oct 2021 14:30:59 +0200

> I don't see how using comments helps anything. The Oracle case makes it pretty clear an API itself can be covered by a copyright, and surely the comments are covered by the copyright.

1) It seems to me that you are confusing the copyright owner with the right to use the interface. Undoubtedly, ISO is the copyright owner. But they may authorize unlimited use of the specification, otherwise NO standard would make sense. Do you infringe copyright if you build an electrical plug that conforms to your electrical standard?

2) Comments help, because they describe precisely what is expected by every function, and what it provides. Actually, I never open the ASIS standard, everything I need is detailed in the comments.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 13 Oct 2021 20:48:11 -0500

> 1) It seems to me that you are confusing the copyright owner with the right to use the interface.

That's clearly covered by "fair use". But API Standards are different: you have to copy large parts of the Standard to implement them (and ASIS is an extreme case -- you have to copy 90% of it to use it). That certainly is not covered by "fair use".

It's my (semi-informed) opinion that API Standards are useless, because you have to violate the ISO copyright to use them (or buy a license).

> 2) Comments help, because they describe precisely what is expected by every function, and what it provides. Actually, I never open the ASIS standard, everything I need is detailed in the comments.

Exactly. Someone copied 90% of the ASIS standard without permission, and *that* is what you are using. And that is depriving ISO of possible revenue.

It's clear to me that anyone using ASIS specs is skating on thin ice. Whether it ever would become a problem for ISO is certainly unknown, but I wouldn't want to build a business on top of such a thing. It's definitely not open source by any reasonable definition.

We've spent a huge amount of effort to ensure that the Ada language (and its language-defined packages) do not fall into the same trap. But it's way too late to do that for ASIS.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 14 Oct 2021 08:09:26 +0200

> It's my (semi-informed) opinion that API Standards are useless, because you have to violate the ISO copyright to use them (or buy a license).

Standards are meant to be used. Therefore my not-better-informed opinion is that the problem has been addressed by ISO, with a decision that APIs, as defined in the standard, can be used.

> Exactly. Someone copied 90% of the ASIS standard without permission, and *that* is what you are using. And that is depriving ISO of possible revenue.

Not at all. The exact specification of ASIS packages is part of the standard, including comments. And this standard has been approved by ISO, with comments.

Ada-related Tools

Simple Components v4.57

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v4.57
Date: Sun, 11 Jul 2021 13:40:44 +0200
Newsgroups: comp.lang.ada

The current version 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, multiple connections server/client designing tools and protocols implementations. The library is kept conform to the Ada 95, Ada 2005, Ada 2012 language standards.

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

Changes to the previous version:

- Bug fix in the HTTP client. The bug affected systems with above average performance causing sporadic distortion of HTTP request headers.

SweetAda on Github

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda on github
Date: Fri, 30 Jul 2021 16:52:52 -0700
Newsgroups: comp.lang.ada

SweetAda has now a home in GitHub.

You can reach it @ https://github.com/gabriele-galeotti/SweetAda.

SweetAda is a lightweight development framework to create Ada systems on a wide range of machines. Please refer to https://www.sweetada.org.

SweetAda is now licensed under the terms of the MIT license. RTS and LibGCC files keep their original license, which is a GCC runtime library exception 3.1.

I've also built a new toolchain release, based on GCC 11.1.0, which will be uploaded in the next days both at SourceForge and SweetAda.org.

The committed branch has some minor changes from the last v0.8 package, and new interesting features, like the possibility to compile the RTS directly from sources, and a fully usability in an MSYS2 environment (for MSYS2 first download the new toolchain and be sure to select only the items you're interested in, because the build script is querying the Makefile and is very slow under that environment).

Yet I have very scarce time, and the documentation is thus painfully incomplete. But do not hesitate to ask.

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Date: Tue, 3 Aug 2021 01:46:21 -0700

> Is this with the generic-instantiation exception, or am I thinking of a different license?

RTS source files and some LibGCC assembly files are, more or less, exact copies of the FSF GCC release, plus some patches. So I've reported their licenses as highlighted in their headers:

[Extract of GPL v3 omitted. —arm]

I'm not a lawyer, and I don't want to hurt anyone, so I've just tried to stay in a "maximum correctness mode", reporting licenses verbatim.

But I think that the whole SweetAda hierarchy, due to this, is practically under the MIT license, and has no limitations.

Corrections welcome.

> How integral is MSYS2 to everything?

SweetAda does work in a windoz environment just in plain cmd shell (with the aid of PowerShell), because the package includes a port of make, grep and sed utilities.

MSYS2 (or Cygwin), plus the dos2unix utility, is required only to rebuild the RTS, because the script is currently Bash-only. So if you are a windoz guy and you want to use a clone from the github repository, you need it. The bad news: MSYS2 is extremely slow in processing scripts.

Obviously SweetAda works much better in a Linux environment, because this is my native environment. OS X should work ok, but it is increasingly difficult for me to make toolchains in that environment (there are problems indeed), and I am limited to checking things in a VM-hosted machine.

SweetAda Update to GCC 11.1.0

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda toolchains updated, GCC 11.1.0
Date: Wed, 4 Aug 2021 10:37:51 -0700
Newsgroups: comp.lang.ada

Hi all.

I've just released an updated version of SweetAda toolchains.

SweetAda is a lightweight development framework to create Ada systems on a wide range of machines. Please refer to https://www.sweetada.org.

The new toolchain is based on GCC 11.1.0. Binutils is @ 2.37 for Linux, still @ 2.35 for Windows and OS X. GDB is @ 10.2.

You can find the toolchains at both SweetAda home, or at SourceForge SweetAda repository https://sourceforge.net/projects/sweetada/files/toolchains. Please browse into [Linux|Windows|OSX]/release-20210725 subdirectories. OSX toolchains in SourceForge are uploading, ready made in sweetada.org.

The new toolchain is supported by SweetAda GitHub repository code.

For older toolchains, you have to revert the RTS GCC 11.1.0 commit (d40b4d0)c61bc901b8d57e16dccb6857fc4182adf, because new long integer types were introduced.

Gnu Emacs Ada Mode v7.1.6

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Gnu Emacs Ada mode 7.1.6 released.
Date: Sat, 31 Jul 2021 09:43:49 -0700
Newsgroups: comp.lang.ada

Gnu Emacs Ada mode 7.1.6 is now available in GNU ELPA.

This is a bug fix release.

ada-mode and wisi are now compatible with GNAT FSF 11, Pro 22, Community 2021.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 5 Aug 2021 10:16:36 +0200

Thank you for the update Stephen.

I can confirm that it works with GNAT community 2021. It is the first time I got it working.

However, a couple of comments. I could not use the build.sh script directly. The line
WISI_DIR=`ls -d ../wisi*`
matches everything named wisi. In my case it is the following:
../wisi-3.1.5
../wisitoken-grammar-mode-1.2.0
../wisi-3.1.5.signed
../wisitoken-grammar-mode-1.2.0.signed

The *.signed entries are files, which screw with gprclean/build. So I had to run each command manually. That is not a big issue for me, but as you can understand, it can easily be a headache to a lot of people. For that reason I would like to propose a small change. The wisitoken package uses a slightly modified ls command:

export GPR_PROJECT_PATH=
`ls -d ../wisi-3.1.?`

This type of pattern matching would solve the issue of finding too many things. I would change the WISI_DIR entry to

WISI_DIR=`ls -d ../wisi-?.?.?`

Which ensures that only the directory of the package is matched. As far as I know, the ? pattern matching is POSIX compliant, so it should work in pretty much any $SHELL.

The second proposal would be to, instead of asking the user to run the commands directly, that an elisp function is used. For example, the package pdf-tools requires some compilation in order to use it. It comes with the elisp function

(pdf-tools-install)

which installs the package, and it is pretty obvious to the user. It also comes with

(pdf-loader-install)

which is recommended to be used in the configuration file. This function checks whether the package has already been compiled/installed properly at boot. If it is, then Emacs just loads it, if not, it gets compiled. I think this is a much more user friendly experience, it would simplify the installation process.

In the case of ada-mode, if the compilation fails, some report could be emitted, such as "No Ada compiler found.", "The GNATCOLL dependencies have to be installed previous to the compilation, please refer to XXX." and finally "Compilation failed, please see the compilation window."/"Compilation successful.". A function the style

(load-or-install-ada-mode)

could be created for this and the user could be requested to run it. Or when ada-mode is called, that function could be executed. That way even if the user is not aware of the compilation step, they are informed. Instead of getting a cryptic wisi error message.

Thank you for your work and fixing the issue with the newer versions of the compilers!

From: Simon Wright <simon@pushface.org>
Date: Thu, 05 Aug 2021 09:58:08 +0100

> I can confirm that it works with GNAT community 2021. It is the first time I got it working.

In my case it didn't: resurrection of old problems,

```
sal-gen_unbounded_definite_stacks.adb:
209:07: error: access discriminant in
return object would be a dangling
reference
sal-gen_unbounded_definite_stacks.adb:
216:07: error: access discriminant in
return object would be a dangling
reference
wisitoken-syntax_trees.ads:620:04:
warning: in instantiation at
sal-gen_unbounded_definite_vectors.adb
:65 [-gnatwv]
wisitoken-syntax_trees.ads:620:04:
warning: aggregate not fully initialized
[-gnatwv]
etc etc
```

This is because ada-mode-7.1.6 only "requires" wisi-3.1.3, which was already installed, rather than the latest 3.1.5 ... install 3.1.5, *delete 3.1.3*

> I could not use the build.sh script directly. The line

>> WISI_DIR=`ls -d ../wisi*`

> matches everything named wisi. In my case it is the following:

> ../wisi-3.1.5   ../wisitoken-grammar-mode-1.2.0

> ../wisi-3.1.5.signed ../wisitoken-grammar-mode-1.2.0.signed

I used

WISI_DIR=`ls -d ../wisi* | grep -v signed`

but Fernando's suggestion is better, I think.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 5 Aug 2021 12:59:53 +0200

> This is because ada-mode-7.1.6 only "requires" wisi-3.1.3, which was already installed

I would like to point out that I had never installed or compiled any of the components. Previous to this successful installation, I updated _all_ packages. Which yielded the newest version of ada-mode and wisi.

I am just giving a bit more information and context, in case someone tries to replicate what I did.

From: Simon Wright <simon@pushface.org>
Date: Thu, 05 Aug 2021 14:42:31 +0100

I got a successful build with macOS GNAT CE 2021. However, using it failed with

```
Execution of /Users/simon/.emacs.d/elpa/
ada-mode-7.1.6/gpr_query terminated by
unhandled exception
raised PROGRAM_ERROR :
gnatcoll-sql_impl.adb:198 accessibility
check failed
Load address: 0x1066e1000
```

[Traceback addresses omitted. —arm]

which is an unfortunate bleeding-edge interaction between the compiler and gnatcoll-db:v21.0.0 (there are mutterings in the source at this line about "GNAT bug OB03-009").

Building with FSF GCC 11.1.0 is looking good.

Which version of gnatcoll-db did you use, Fernando? The ada-mode README isn't very prescriptive.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 5 Aug 2021 15:51:36 +0200

> Which version of gnatcoll-db did you use, Fernando? The ada-mode README isn't very prescriptive.

I used the master branch as of two days ago, so 2021/08/03. I thought about downloading a tagged version, as of that day the 21.0.0. But since I had already cloned master, I went with it.

I think this issue is related to this commit, which refers to that GNATbug directly:

https://github.com/AdaCore/gnatcoll-db/commit/c75234037fb4568739435fad204f206afe609a77

From: Manuel Gomez <mgrojo@gmail.com>
Date: Sat, 7 Aug 2021 00:00:19 +0200

I share my experience, just in case it's useful for anyone.

I tried with GNAT CE 2021 and gnatcoll-db 2021, but had problems with that combination. I finally used gnat-9 and gnatcoll packages provided by Ubuntu 20.04, but had to add ".all" for "error: access discriminant in return object..." and remove the option to consider warnings as errors in "standard_common.gpr". Different compiler versions produce different warnings, so that flag might be counterproductive for distribution. The problem with the 2021 version might have been the same, not sure about it.

From: Simon Wright <simon@pushface.org>
Date: Sat, 07 Aug 2021 11:35:19 +0100

> Previous to this successful installation, I updated _all_ packages. Which yielded the newest version of ada-mode and wisi.

Thanks for the advice to update all packages (U x in the package list window), which gives us ada-mode-7.1.7, which includes the "ls -d wisi*" fix, and builds/works fine on macOS Big Sur with GCC 11.1.0 (see previous remarks re: gnatcoll-db).

From: Paul Onions <ponions37@gmail.com>
Date: Sat, 7 Aug 2021 09:47:38 -0700

> ... builds/works fine on macOS Big Sur with GCC 11.1.0 ...

I'm using the same setup now, but for some reason I'm still seeing error messages about a void-function called wisi--lexer-error. I can get a working system if I go into the wisi-3.1.5 directory and delete all of the .elc files I find there, but then editing can be very slow (e.g. delay between pressing a key and character appearing in buffer >5 secs sometimes). Not sure if this is caused by my deleting the .elc files, but it also happened to me when I did the same thing in my workarounds to get the 7.1.4 ada-mode release working.

From: Paul Onions <ponions37@gmail.com>
Date: Sat, 7 Aug 2021 10:46:00 -0700

> ... I'm still seeing error messages about a void-function called wisi--lexer-error.

I think this is due to a missing:-

```
 (require 'cl-lib)
```

at the start of wisi-parse-common.el.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 09 Aug 2021 03:54:21 -0700

> [...] The line

> WISI_DIR=`ls -d ../wisi*`

> matches everything named wisi.

Fixed in 7.1.7

> The second proposal would be to, instead of asking the user to run the commands directly, that an elisp function is used.

I keep hoping *someone else* will implement this :).

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 09 Aug 2021 14:09:47 -0700

> In my case it didn't: resurrection of old problems,

> This is because ada-mode-7.1.6 only "requires" wisi-3.1.3,

Sigh. to be fixed in 7.1.8.

I could argue that wisi 3.1.3 is technically correct, since ada-mode 7.1.6 does compile with it. But that's not really the point here; easy user upgrade is more important.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 09 Aug 2021 14:18:41 -0700

> I got a successful build with macOS GNAT CE 2021. However, using it failed [...]

tried to attach a patch for that, but nntp.aioe.org says "441 Invalid Content type" even for text/plain. So I include it inline below; it applies to the 21.2 release branch of gnatcoll-sql from github AdaCore. I guess I should have included it in 3.1.8.

> Which version of gnatcoll-db did you use, Fernando? The ada-mode README isn't very prescriptive.

ada-mode.info has more detail in the Install section.

-------------- gnatcoll-2021-sql.patch -------

```
--- sql/gnatcoll-sql_impl.adb.orig
   2021-05-20 01:25:55.000000000 -0700
+++ sql/gnatcoll-sql_impl.adb
   2021-06-21 09:44:09.437292100 -0700
@@ -188,15 +188,9 @@
(Self : Field;
To : in out SQL_Field_List'Class;
Is_Aggregate : in out Boolean)
- is
- FC : access SQL_Field_Internal'Class;
- begin
+ is begin
if not Self.Data.Is_Null then
- --  !!! Could not use Element call result in
the
- --  Append_If_Not_Aggregate parameter
because of GNAT bug OB03-009
-
- FC := Self.Data.Get.Element;
- Append_If_Not_Aggregate (FC, To,
Is_Aggregate);
+ Append_If_Not_Aggregate
(Self.Data.Get.Element, To, Is_Aggregate);
end if;
end Append_If_Not_Aggregate;
end Data_Fields;
--------------------------
```

GNAT CE 2021 for Intel MacOS

From: Simon Wright <simon@pushface.org>
Subject: Re: ANN: GNAT CE 2021 for Intel macOS
Date: Wed, 18 Aug 2021 17:11:27 +0100
Newsgroups: comp.lang.ada

GNAT CE 2021, built for macOS El Capitan .. Big Sur, updated.

The problems addressed are:

* Bad dylib path in libxmlada_unicode.dylib.2021: I hadn't cleared out a previous build attempt: fixed.

* Gnatcoll.Xref crash: this also affected emacs ada-mode: patched.

* GDB `file` command crash: patched.

Additionally, note:

* The Gnatcoll Python binding is (a) to Python 3, (b) to the https://python.org release, which doesn't install in the same place as the Big Sur Xcode release.

* Only the Sqlite backend for Gnatcoll DB is provided.

Sourceforge: https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2021-x86_64-darwin-bin-2/

Github (scroll down to the Assets section): https://github.com/simonjwright/distributing-gcc/releases/tag/gnat-ce-2021-2

GtkAda on MacOS Big Sur

From: Gareth Baker <garethbaker60@gmail.com>
Subject: GtkAda on macOS Big Sur
Date: Sat, 21 Aug 2021 07:28:29 -0700
Newsgroups: comp.lang.ada

I hope someone can help - I've used GtkAda before with no problems (AdaCore CE and Xnadalib) but I'm now getting an odd behaviour. Using the recent CE2021 versions (and actually all other previous versions back to 2019). The programs compile okay but when launched they appear as a small rectangle (top left quarter) within a larger window with a black background.

Am I missing some setting that is not mentioned in the README(s)?

From: Jeffrey R. Carter
Date: Sun, 22 Aug 2021 11:14:05 +0200

It sounds as if the OS version changed something. You might want to consider a less OS-dependent GUI library, such as Ada GUI (https://github.com/jrcarter/Ada_GUI) or Gnoga (https://sourceforge.net/projects/gnoga/).

From: Gareth Baker <garethbaker60@gmail.com>
Date: Fri, 27 Aug 2021 09:13:42 -0700

The testgtk program does the same thing, a window opens up with the program shrunk to 1/4 size against a black background.

The gtk3-demo is slightly different in that it opens up on its own but I think again it is 1/4 of the size it should be and the mouse clicks do not work where they should.

I'm on macOS 15.5.2.

Other programs run from the terminal (python with qt GUI) work okay.

From: Simon Wright <simon@pushface.org>
Date: Fri, 27 Aug 2021 18:12:27 +0100

> The testgtk program does the same thing [...] The gtk3-demo is slightly different

Same here.

> I'm on macOS 15.5.2.

11.5.2 I think!

I had to run "sudo xattr -d com.apple.quarantine" on bin/*, lib/*.dylib, lib/*.so, and the testgtk program.

Also, on page 2 of https://blady.pagesperso-orange.fr/telechargements/gtkada/gtk-ada.pdf,

it should say

```
$ PATH=/opt/gnat-ce-2021/bin:$PATH
```

not

```
$ PATH=/opt/gnat-ce-2021:$PATH
```

From: Gareth Baker <garethbaker60@gmail.com>
Date: Fri, 27 Aug 2021 11:58:06 -0700

Humm - not sure why but removing the security attribute does not work for me.

From: Simon Wright <simon@pushface.org>
Date: Fri, 27 Aug 2021 20:48:39 +0100

Without removing it the programs won't run at all, unless you've turned off system integrity protection (bad idea).

And having removed it, as I said, I see the same unexpected behaviour you do.

Pascal, I'm on a MacBook Pro (Retina, 13-inch, Early 2015)

Emacs Mode: Using Tree-sitter

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Subject: Emacs mode: using tree-sitter
Date: Mon, 30 Aug 2021 02:21:47 -0700
Newsgroups: comp.lang.ada

I was looking recently at both Emacs and vim recent updates, and noted that both those tools now provide interfaces to tree-sitter (https://tree-sitter.github.io/tree-sitter/) which is a parser generator and incremental parsing library. It doesn't have an Ada parser yet, though :-(

It might be nice, as a community, to work on such a parser though. I did not look into what that implies yet, maybe someone else has already started work on that.

The advantage might be that the Emacs ada-mode can use that instead of its home-brewed parser (which although I am sure it was fun to develop still likely requires some maintenance by Stephen, and definitely requires manually compiling some Ada code before we can use the ada-mode).

We could also use it to improve the current vim ada-mode, which hasn't been updated in years and could do with various improvements.

Finally, maybe we could talk with the GNAT Studio team. I don't think they have looked into tree-sitter yet, but it might be useful.

I do not know whether tools like Visual Studio Code also interface with tree-sitter, but maybe that library will become the equivalent of the Language Server Protocol, and companies provide one tree-sitter parser + one language server and the IDE automatically gains support for Ada

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 30 Aug 2021 17:37:58 -0700

> [...] It doesn't have an Ada parser yet, though :-(

And it won't until the parser generator gets a serious overhaul: https://github.com/tree-sitter/tree-sitter/issues/693

> [...] maybe someone else has already started work on that.

Yes, me. There is code in wisitoken devel that converts any wisitoken grammar to tree-sitter syntax. I find EBNF much more readable than the javascript DSL tree-sitter uses.

> The advantage might be that the Emacs ada-mode can use that instead of its home-brewed parser [...]

Someone would have to maintain the tree-sitter parser; it's not magic. And you'd have to compile the tree-sitter parser as well. Again, not magic.

> We could also use it to improve the current vim ada-mode

It might be easier to adapt the Emacs ada-mode code to meet the vim plugin interface. Or adapt Emacs ada-mode code to LSP, that would benefit many editors.

> Finally, maybe we could talk with the GNAT Studio team. I don't think they have looked into tree-sitter yet, but it might be useful.

They provide an LSP ada-language-server: https://github.com/AdaCore/ada_language_server

It is not as feature rich as the ada-mode parser.

I doubt they have the resources for anything more (unless the request comes with money, of course).

Adare_net Ada Network Initial Release

From: Daniel Norte Moraes <danielcheagle@gmail.com>
Subject: ANN: Adare_net Ada network lib
Date: Sat, 4 Sep 2021 20:54:53 -0700
Newsgroups: comp.lang.ada

Hi All! :-)

Adare_net is a small, portable and easy to use Ada network lib. It supports ipv4 ipv6 udp and tcp, and can 'listen' with ipv6, too.

The powerful buffer feature can support all Ada types, and with a more refined treatment, you can use endian proof records and unconstrained arrays.

From now, tested and working:

AMD64 : MSWindows 7 sp1 64bits and Ubuntu Hirsute 64bits

Thanks and Enjoy!!

https://gitlab.com/daresoft/network/adare_net

From: Drpi <314@drpi.fr>
Date: Fri, 17 Sep 2021 23:04:58 +0200

I had a quick look at the top level source code. I'm surprised that all packages are declared with "pure" aspect. From what I understand of the "pure" aspect, these packages are not pure. Am I wrong?

From: Joakim Strandberg <joakimds@kth.se>
Date: Wed, 22 Sep 2021 01:47:13 -0700

I agree with you Nicolas, they should not be declared Pure. It makes the GNAT compiler check for example that there are no global variables used in the packages but other than that, they (I didn't check all the packages) are not Pure. The pragma Pure worked as expected in Ada83 but the meaning and utility of it disappeared with the Ada95 standard. [...]

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Sep 2021 02:16:12 -0700

There is no pragma Pure in Ada 83.

From: Joakim Strandberg <joakimds@kth.se>
Date: Wed, 22 Sep 2021 04:07:05 -0700

Thanks for clearing that up AdaMagica, I wasn't aware.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 27 Sep 2021 23:52:33 -0500

>There is no pragma Pure in Ada 83.

Pragma Pure was an IMHO failed attempt to control/document access to globals. It has much too broad of a granularity to be very useful (I've never found anything that I could make Pure outside of language-defined things, and some of those cannot be implemented as Pure even though declared that way). Ada 2022 has aspect Global to do this properly, Global => null has many fewer holes than Pure.

Note however that one can always lie about any Ada semantics in interfacing code. But any such lies make your code erroneous, and while it might work on one compiler today, there's no guarantee that it will work anywhere else (including the next update of your usual compiler). See B.1(38.1/5):

"It is the programmer's responsibility to ensure that the use of interfacing aspects does not violate Ada semantics; otherwise, program execution is erroneous. For example, passing an object with mode "in" to imported code that modifies it causes erroneous execution. Similarly, calling an imported subprogram that is not pure from a pure package causes erroneous execution."

(The latter two sentences were added because programmers didn't seem to get what the first sentence means. We wanted that to be interpreted in the broadest possible way.)

AdaControl v1.22r15

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] New version of AdaControl released
Date: Wed, 22 Sep 2021 22:42:12 +0200
Newsgroups: comp.lang.ada

Adalog is pleased to announce a new version of AdaControl (1.22r15).

This version features a number of new rules and enhancements, reaching 73 rules and 591 possible checks.

Noteworthy improvements include a rule to check for known exceptions; this includes a data-flow tracing function, that benefits other rules too; a subrule to check assignments that could benefit from the new "@" syntax of Ada 202X, and other simplifiable statements; enhanced detection of redundant instantiations of generics, and more.

There is also a possibility to define your own output format, with examples using Toml and Yaml formats.

As usual, the complete list of improvements and new features can be found in file HISTORY.

Installation procedures have slightly changed, due to AdaCore's decision to not provide the community with the useful tools that it reserves to paying customers. Please read the details on AdaControl's home page, where you can download this version from:

https://www.adacontrol.fr

Enjoy!

Ada-related Products

Janus/Ada 1.5 CP/M Manuals

From: Luke A. Guest <laguest@archeia.com>
Subject: Janus Ada 1.5 Ada cp/m manuals
Date: Sun, 15 Aug 2021 00:05:34 +0100
Newsgroups: comp.lang.ada

Does anyone have electronic copies? They're not in the zips available.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 17 Aug 2021 02:23:43 -0500

Unfortunately, they don't exist. The original source was for a photo-typesetter that hasn't existed for decades (and most likely is only stored on 8" floppies that probably aren't readable even if the right hardware was available).

The only way for them to exist is for someone to scan a printed version.

I have a single printed version in our archives (with installation instructions for 8" floppies dated March 5, 1984). It's got someone's handwritten notes in it (it's not pristine). Anyway, since it is the only known version, I won't let it out of the office, since it is literally irreplaceable.

I've offered to others to scan it to PDFs (one per page, that's all I can figure out how to do on our cheap multifunction machine here) if someone would want to convert that to something more usable. But since I'd have to scan each page individually and the document is an inch thick give or take a few millimeters, I'd need some compensation for the time. (I have lots of things I could be doing that are either fun, make money, or advance Ada - this is none of these!) Contact me privately if you want to discuss this further.

BTW, since RRS still exists and never made a public version for any version of Janus/Ada (including the CP/M versions) -- because all of the versions are derived from the same original source and it is likely that some of the compiler still survives from those versions) -- using it without a license is technically infringing. I don't think there is much chance that anyone would try to stop non-commercial use, but I would suggest getting legal if anything commercial is involved. (And yes, we periodically get requests for help with it from users that I would have expected to have moved on long ago.)

From: Paul Rubin <no.email@nospam.invalid>
Date: Tue, 17 Aug 2021 02:03:43 -0700

> I've offered to others to scan it to PDFs (one per page, that's all I can figure out how to do on our cheap multifunction machine here)

Do you think you could scan and post one page, maybe from the middle, so we would know what we are dealing with?

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 17 Aug 2021 10:16:22 +0100

> [...] that probably aren't readable even if the right hardware was available.

There are a few retro youtube channels who could handle getting any old hardware working if you have it and are willing to donate or lend it to them, retro recipes, rmc, 8-bit guy, etc.

> I've offered to others to scan it to PDFs (one per page, that's all I can figure out how to do on our cheap multifunction machine here) if someone

Maybe you should let someone come in with their own laptop and scanner to do it.

> BTW, since RRS still exists and never made a public version for any version of Janus/Ada (including the CP/M versions) [...]

Seriously? Even Caldera released CP/M to the public and Amstrad released the source to the Spectrum ROM's.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 17 Aug 2021 11:25:54 +0200

I saw a few forums about reading 8" floppies. It looks quite doable. Floppies are very reliable too, my old 3.5" floppies from the 90's are still readable. I think there are good chances of success.

Conversion from typesetting to HTML would be a neat Ada program... (:-))

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 17 Aug 2021 10:28:02 +0100

> Seriously? Even Caldera released CP/M to the public and Amstrad released the source to the Spectrum ROM's.

Having an official public release would be good for historical reasons. Having the source would be even better, for curios like me who have been wondering how old 8-bit compilers worked.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 18 Aug 2021 15:05:29 -0500

> Maybe you should let someone come in with their own laptop and scanner to do it.

That would be an option; I didn't think of it as the last person that wanted it was in Scandinavia and visiting Madison WI would be far more expensive than giving me a few hundred dollars to do it. If there is some US-based person that wants to do that, the dynamics are different.

>Having an official public release would be good for historical reasons.

Most likely don't have the capability to make such a release (I do have a Z-80 CP/M machine in storage, but it's unlikely to boot - S-100 machines stored a year usually needed extensive cleaning of contacts to work, after 20 years...).

> Having the source would be even better, for curios like me who have been wondering how old 8-bit compilers worked.

So far as I know, that's (partially) lost. I had moved it to dual 5 1/4" floppies, and I was asked to throw out all of the redundant stuff to save space when we moved to a smaller space. When I was retiring the last working machine with 5 1/4", I decided to move it into our version control, but was unable to read all of the floppies. So parts are lost. That's OK from an RRS perspective, as we wouldn't use an ancient code generator in anything new anyway (it would need to hook to the modern optimizer/static analyzer, so it would need a full redo anyway). I did manage to get the runtime into our version control, something that would be a lot more work to reproduce.

In any case, parts of the source (and more importantly, design) are still in use so giving it away isn't really an option.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 18 Aug 2021 21:34:14 -0500

> Do you think you could scan and post one page, maybe from the middle, so we would know what we are dealing with?

That I can do. See http://www.rrsoftware.com/archives/CPM-doc-sample.zip

There are three pages in here, one from the 3.3 upgrade text, and two from the regular 3.2 printed manual. (I don't think there ever was a consolidated version.)

Looking at this in more detail, it looks like the original documentation was printed on the NEC Spinwriter with a special font and ribbon. The formatting program was something we built, it was related to the typesetter version but that might have been a bit later.

The 3.3 update seems to have been printed on a lousy dot matrix printer, probably whatever we were using at the time. That's why I scanned a sample of each. The 3.3 update seems to be an MS-DOS version, unfortunately (it talks about 8087 at one point); for the most part those were the same but there is probably some CP/M specific stuff that's missing.

I haven't been able to get the HP scanning software to work reliably on our network, so I have to scan each page individually and e-mail it to myself. That works fine for things that are just a single page (like invoices) but gets really old for a large document. (There are about eight steps on the tiny touch screen of the printer for each page.) Not sure that it would be much easier even with the software, because I'd still have to go into the other room and change each page to be scanned (no feeder on this scanner).

From: Paul Rubin <no.email@nospam.invalid>
Date: Thu, 19 Aug 2021 22:25:41 -0700

> room and change each page to be scanned (no feeder on this scanner).

It might be easier to just photograph all the pages with a phone. That would be harder to OCR than good quality scans, but at least it would preserve the contents.

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 20 Aug 2021 00:58:06 -0700

> it looks like the original documentation was printed on the NEC Spinwriter... The 3.3 update seems to have been printed on a lousy dot matrix printer,

Any idea of the total number of pages involved The Spinwriter text is quite readable and probably OCR-able, not counting the handwritten notes. The dot matrix update would be more difficult.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 24 Aug 2021 18:27:56 -0500

> Any idea of the total number of pages involved?

I can't easily tell; the pages are numbered by chapter (2-3, 4-5, 8-4, etc.). It's about 19 mm (using a ruler) of 24 lb paper (we printed these things to last), including 4 heavy dividers. So it's probably not a huge number. I guessed 200+ when I was trying to figure out how much time it would take me to deal with it.) BTW, some of the pages have yellow highlights as well. I don't think that would cause problems for an OCR, but I don't know.

Note that a lot of the manual is a basic outline of Ada; the original manual was created back in the early days of Ada when other material wasn't readily available. But it also includes details and limitations on the various features, which is interesting for a subset. So just skipping that material isn't a great idea.

Ada and Operating Systems

DEC Ada for VAX/VMS 5.5

From: Calliet Gérard <gerard.calliet@pia-sofer.fr>
Subject: dec ada for vax/vms 5.5
Date: Thu, 29 Jul 2021 17:40:44 +0200
Newsgroups: comp.lang.ada

I'm not an archeologist. Somewhere in the world they use VAX/VMS 5.5-H2, and will for a long time. I'll be retired when they stop the servers.

So I'm training new guys who will do the job. I try to help them take the job as fun as possible.

We learn this year DEC Ada

I need:

- a distribution

- buying a license

Anyone know where I can get these things?

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 29 Jul 2021 09:52:31 -0700

Try VMS Software:

https://vmssoftware.com/contact/

They're doing the port of OpenVMS to x86, and have a suite of software they're updating [IIRC they also have existing support contracts for VMS, so might know where to get it]; it would probably be good to call them and tell them your company would like Ada support in the port, too.

(It certainly can't hurt to signal that there's interest there.)

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Thu, 29 Jul 2021 13:40:13 -0400

https://training.vmssoftware.com/student-license/

https://vmssoftware.com/community/community-license/

No idea if the Ada compiler is installed. My prior employer was running a version of VMS on some Windows server, no doubt via emulator -- after replacing the ancient VAX systems. It was needed to support an ancient (non-DEC) Ada cross compiler for 680xx processors. (Yes, the Boeing* 737 flies on 68040 CPUs <G>)

https://blog.poggs.com/2020/04/21/openvms-on-a-raspberry-pi/

Unfortunately, I think /that/ free version of OpenVMS has been discontinued.

As you have commented, the hobbyist license program was terminated last year and many people are now unable to use any system which required LMF (VMS 5.0 and above).

http://www.vaxhaven.com/CD_Image_Archive

But doesn't provide licenses to activate...

* Boeing is NOT the employer in question

From: Calliet Gérard <gerard.calliet@pia-sofer.fr>
Date: Fri, 30 Jul 2021 16:22:46 +0200

> Try VMS Software

VSI doesn't do anything for VAX environments (alpha, itanium, x86 (soon), yes, VAX, no). And HPE seems to have totally abandoned VMS.

> They're doing the port of OpenVMS to x86 [...]; it would probably be good to call them and tell them your company would like Ada support in the port, too.

THE Big Issue. I'm calling them since 2015, some others are calling them now. They hesitate. I think because the ratio: number of business cases / investment is not so good.

(I'm myself involved on Ada on VMS: www.vmsadaall.org)

> (It certainly can't hurt to signal that there's interest there.)

Right

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Fri, 30 Jul 2021 11:35:15 -0700

> We learn this year Dec Ada

> I need:

> - a distribution

> - buying a license

You might consider contacting AdaCore (sales@adacore.com I believe). I remember that quite a number of things were added to the GNAT compiler for compatibility with DEC Ada. The biggest difficulty is likely to be the build process (and we have related discussions on some other thread in this forum), because DEC Ada's notion of systems and subsystems is quite different from what GNAT does. But AdaCore has some experience on the subject.

Porting Ada to NetBSD

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: Help: Ada in NetBSD
Date: Sun, 29 Aug 2021 13:06:53 +0200
Newsgroups: comp.lang.ada

Dear All,

I have been trying for the past few months to make GCC/Ada work in NetBSD. I am writing this message to you since I have been stuck in a roadblock for far too long and without concrete answers.

Long story short: JMarino, within the Aurora project, already ported GCC/Ada to a lot of systems, namely FreeBSD, DragonflyBSD, NetBSD and Solaris. The last version that works without friction in NetBSD/pkgsrc is GCC v6. I wanted to update GCC to v10 (10.3.0).

So, one can compile GCC v10 with C, C++ and Ada support with v6 without any issues. The biggest problem is that the RT (RunTime Files) had no configuration for NetBSD (see the original Makefile.rtl in the gcc/ada directory). I fixed it by copying the FreeBSD support files and modifying an imported C function to be POSIX compliant, since NetBSD did not have the function that FreeBSD used (related to pthreads).

The results of compiling GCC v10 with this "small" change are documented in a blog entry I did:

https://www.irvise.xyz/Projects %20&%20Engineering/ updating-gcc-ada-pkgsrc.html

TL;DR: GCC v10 compiles and can generate binaries!!! :D But...

The tasking system is not working correctly (I have been testing the compiler with the ACATS test suite provided by Simon). The linker complains about some C functions not being correctly imported within Ada files. And the programs where the linker complains, once compiled, tend to get blocked or die. Here is one such example:

/usr/bin/ld:

/home/fernando/mysandboxfolder/usr/ pkg/gcc10/lib/gcc/ x86_64--netbsd/10.3.0/adalib/ libgnat.a(s-osprim.o):

in function `system__os_primitives__clock':

/usr/pkgsrc/wip/gcc10-aux/ work/build/gcc/ada/rts/s-osprim.adb:91: warning: reference to compatibility gettimeofday(); include <sys/time.h> to generate correct reference

As you can see, the linker says that, in this case, gettimeofday() is not correctly referenced and that I should include <sys/time.h>. Notice, it is complaining that the file s-osprim.adb, and Ada file, is at fault here. This happens to all files that use the tasking system in one way or another, so, in summary, all large projects, such as GPRBuild.

I thought that an #include <sys/time.h> may have been missing from a C source file that is required to build the Ada compiler. After all, there were some defined (__NetBSD__) missing from the Ada sources.

I added those. Nothing. I took a really good look at JMarino's patches:

http://cvsweb.netbsd.org/bsdweb.cgi/ pkgsrc/lang/gcc6-aux/files/ diff-ada?rev=1.1&content-type=text/ x-cvsweb-markup

I applied some extra changes (the configure/configure.ac patches are failing to apply). Still nothing, it keeps failing.

I have been looking for the "missing" #include files, they are <time.h>, <sys/time.h> and <signal.h>. I searched through the code, there are few occurrences of them and, for example, <sys/time.h> only appears in a legacy system.

I checked the C signature files to make sure that they were also correct in the Ada sources, and they seem to match.

I am out of ideas.

How come the linker complains about those functions and not the other imported C ones? These files are automatically included with -lc. How could I go about fixing this issue? Any ideas, pointers?

Below* are the patches that I have created.

If you are wondering why am I doing this: I like alternative systems, Ada is portable on paper, but what about in reality? And my end goal would be to see Ada everywhere and upstream these fixes to GCC.

*[For the large amounts of code in this thread, visit https://groups.google.com/ g/comp.lang.ada/c/XXAQEbMsEUM —arm]

From: Stephane Carrez
<stephane.carrez@gmail.com>
Date: Sun, 29 Aug 2021 06:19:57 -0700

On NetBSD there are several symbols that are replaced by the virtue of including a C header. If you include correctly the C header, the correct symbol is used and you don't get the linker warning.

For gettimeofday the symbol is replaced by __gettimeofday50.

These symbols are marked with __RENAME(XXX) macros in the C headers.

I would suggest to have a look at the .o files to find out the one that has the `gettimeofday` symbol that is not replaced.

By the way, I'm interested in your work as I'm still stuck on gcc 6 for my NetBSD machines. 20 years ago I wrote the 68HC11 port that was integrated in GCC 3.3 so I have some past experience in working with GCC. Despite my very limited spare time, I could have a look if you provide me with the sources of your port :-)

From: Simon Wright
<simon@pushface.org>
Date: Sun, 29 Aug 2021 18:34:50 +0100

> The tasking system is not working correctly (I have been testing the compiler with the ACATS test suite provided by Simon).

There are several tasking-related (CXD) tests in ACATS that few if any desktop OSs are expected to support; mainly, I think, priority-related.

[...]

From: Fernando Oleo Blanco
<irvise_ml@irvise.xyz>
Date: Sun, 29 Aug 2021 20:08:27 +0200

> On NetBSD there are several symbols that are replaced by the virtue of including a C header. If you include correctly the C header, the correct symbol is used and you don't get the linker warning.

That is what I did by adding the indicated header files to the NetBSD section of the init.c file. No other systems have them there (or anywhere in some cases). I expected that to fix the issue, but it did not.

> For gettimeofday the symbol is replaced by __gettimeofday50.

> These symbols are marked with __RENAME(XXX) macros in the C headers.

I saw a few of those... So that is what they do... I never got to the bottom of that rabbit hole.

> I would suggest to have a look at the .o files to find out the one that has the `gettimeofday` symbol that is not replaced.

I am doing it right now, let's see what I can find... However, as I said, the headers should have been already included. The linker does not complain during the compilation of gcc. Only when I try to build things with the newly created toolchain. Maybe that is a clue...

> By the way, I'm interested in your work as I'm still stuck on gcc 6 for my NetBSD machines. [...]

I am working directly on pkgsrc/wip. This way I hope to be able to upstream everything as quickly as possible. Jay Patelani already uploaded the patches from the initial blog post. You can find them here:

https://github.com/NetBSD/pkgsrc-wip/ tree/c550eafe889691af212379590974944 e1359e180/gcc10_aux

It is basically the gcc10 entry with the patch-ada* file in patches and Ada added to the USE_LANGUAGES variable (it has to be hardcoded, it cannot be set via options). It is not a clean snapshot, some dirty files were pulled, but it should work as first order approximation. The previous email contains some extra patches (though you would have to update the distinfo file manually). I was lucky that the pkgsrc got changed a few months back to make gcc6-aux the default, instead of gcc5-aux.

I will ask you to take a look if I need to. However, this is "my personal project" I want to do this myself, so for the time being, no need for that :) I would like to see Ada running on as many systems and package managers as possible ;)

P.S: I am yet to try your AWA, I am looking forward to it.

From: Simon Wright <simon@pushface.org>
Date: Sun, 29 Aug 2021 19:25:05 +0100

>> If you include correctly the C header, the correct symbol is used and you don't get the linker warning.

> That is what I did by adding the indicated header files to the NetBSD section of the init.c file. [...] I expected that to fix the issue, but it did not.

The problem can't be fixed by including C headers, because … [...]

The C header is (I got this from the net, so beware)

```
int    gettimeofday(struct timeval * __restrict,
void *__restrict)

   __RENAME(__gettimeofday50);
```

so when you say, in Ada,

```
function gettimeofday
  (tv : not null access struct_timeval;
   tz : struct_timezone_ptr) return Integer;
pragma Import (C, gettimeofday,
  "gettimeofday");
```

the linker looks for a symbol gettimeofday (or maybe _gettimeofday) and gives you the warning that you report. Since it's just a warning, it may actually work - for the moment, anyway.

The Ada needs to change to

```
function gettimeofday
  (tv : not null access struct_timeval;
   tz : struct_timezone_ptr) return Integer;
pragma Import (C, gettimeofday,
  "gettimeofday50");
```

(or maybe "_gettimeofday50", or even "__gettimeofday50" - nm will be your friend).

From: Stephane Carrez <stephane.carrez@gmail.com>
Date: Sun, 29 Aug 2021 15:08:27 -0700

Simon is right, the symbol used by the Ada import statement must be renamed.

The reason for the symbol change is some NetBSD libc change in the signature of some system calls. Some information in: http://ftp.netbsd.org/pub/NetBSD/NetBSD-current/src/lib/libc/README

The __gettimeofday50 is the new function signature while _gettimeofday is the old one. The gettimeofday is the weak alias to _gettimeofday and produces the warning.

Beware that the symbol name that you specify for some import statement is platform specific. Having a different symbol name for NetBSD is quite common.

FreeBSD is different from NetBSD, likewise for OpenBSD :-)

Thanks for the link to the NetBSD git sources, I'm trying to build and keep you informed of my progress :-)

From: Simon Wright <simon@pushface.org>
Date: Mon, 30 Aug 2021 08:37:54 +0100

> Simon is right, the symbol used by the Ada import statement must be renamed.

One possibility, with ample precedent, would be to create e.g. __gnat_gettimeofday() in gcc/ada/adaint.[ch] and reference that in the Ada import statement.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 30 Aug 2021 14:15:18 +0200

Okay. I have a much much clearer picture now.

I have spoken with a couple of people in the NetBSD IRC. NetBSD has been revamping their ABI, see for example, the UNIX time.

Some things were going to break. In the case of some of the "standard" functions, they created a RENAME macro to hide these changes. It leaves a weak symbol reference that is empty and not resolved by the linker.

After taking a much closer look into the patch set of JMarino, I realised that he had already dealt with this issue. For example, see:

https://github.com/NetBSD/pkgsrc/blob/27a8f94efc02f33007d20a4ba6a8aaa369361b95/lang/gcc6-aux/files/diff-ada#L1685

I think I am going to use the strategy that Simon pointed out. Since that would be the most maintainable way for the future... The patching is going to be much longer than I expected.

From: John R. Marino <mfl-commissioner@marino.st>
Date: Wed, 1 Sep 2021 06:28:43 -0700

Fernando,

Maybe you are in luck. A friend of mine needs Ravenports (http://www.ravenports.com/) to support NetBSD. Ravenports has the same base compiler for all supported systems. It was GCC 9.3 but I'm in the process of completing the transition to GCC 11.2.0. This base compiler has to support Ada among other languages. Which is a long way of saying I have to polish my netbsd patches for that compiler and re-bootstrap it back to NetBSD. So I'll be working on this separately (for GCC 11.2, not 10.x).

My process will be different. I cross-compile it on another host, then bring it over to NetBSD and eventually it builds itself natively. I'm awaiting an SSD to arrive which I'll install the latest NetBSD on. My gcc6-aux work has been living on: https://github.com/jrmarino/draco While the patches are current for FreeBSD, DragonFly, Linux, Solaris and probably Android, I did let OpenBSD and NetBSD support slip. But I'm not starting from scratch.

I'll look over your work. And with regard to the test suite, I got all the tests to pass back then:

http://www.dragonlace.net/gnataux/netbsd64/

Which reminds me: I'd only do this for x86_64 platform.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 1 Sep 2021 16:58:47 +0200

[...]

> Which reminds me: I'd only do this for x86_64 platform.

My goal would be to at the very least give support to ARM, ARM64 and RISC-V. To be honest with you, I would like it to work on any piece of hardware that can be currently bought. Also, not just NetBSD, also FreeBSD, DragonflyBSD (already done by you), improved OpenBSD support and Haiku. I would also like to see gcc-ada added to Guix, the GNU package manager, but that is a completely different story.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Fri, 17 Sep 2021 19:36:01 +0200

Another update on my side, this time with a bit more content.

Following the help provided by Arnaud, I modified the flags with which the ACATS's tests get compiled.

To the gnatmake command I added the -f -a -g -j0 flags.

-f to force the recompilation of all files needed with the exception of the runtime and library files.

-a is to also force the recompilation of the library & runtime files.

Whatever is needed.

-g debugging.

-j0 ignored by the ACATS suite provided by Simon (or so says the documentation in the shell file).

These flags make the compilation much slower, obviously, since nearly for every test the entire Ada library needs to be recompiled. However, this started to give me a much better insight on what was going on. Especially since now I could debug the failing tests.

I noticed that the first test I started debugging was stuck in a loop related to task management. This would explain why I was getting so many tests failing with timeouts. Great, but I could only get so much insight.

Arnaud, once again came to the rescue and indicated that I should add the -gnata flag.

-gnata is to enable assertions.

And yes, now the tests were finally failing in a meaningful manner. I have written the assertions I have found that failed. Remember, I am using GCC 10.3 with the patch set that is available on my website. You can find it in one of the messages I have sent in this thread.

So, what do we get?

System.Assertions. Assert_Failure in s-tassta.adb:1643 (very common everywhere), s-tpopmo.adb:213 (fairly common) and s-taprop.adb:463 (common in the c9 section).

Storage_Error in s-intman.adb:136

Stack overflow or erroneous memory access in a few tests. I got no pointer on where it was happening.

And still some timeouts, but I think they are related to the first assertion mentioned.

The "strange" (not that much) is that I have not touched any of these files. I will see where they are used in the compiler, how they are used and if the issues are related with how NetBSD handles some functions/standards... The s-tassta.adb problem I know is 100% related to POSIX Threads. Maybe the issue is in the POSIX Threads handling or maybe not.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 23 Sep 2021 21:53:47 +0200

Okay, so another short blog post. This is going to be a bit of a rant.

So... remember when I said that NetBSD expected a priority value of -1 when using SCHED_ODER? And that that was not POSIX compliant? Well, after a nice conversation in #netbsd, it has been decided to escalate this matter into a PR/ML discussion. All that good :)

But the question on how does Linux work then? Remained... So I ran the ACATS suite with debugging symbols, recompilation and assertions to check. And guess what?

Let the code speak:

```
  else
    Param.sched_priority := 0;
    Result :=pthread_setschedparam
        (T.Common.LL.Thread,
     SCHED_OTHER, Param'Access);
  end if;
  pragma Assert (Result in 0 | EPERM |
     EINVAL);
end Set_Priority;
```

So the Set_Priority function receives the Default_Priority value, which I think was 48. But when it goes into the actual branch, it knows that that default value is stupid and discards it (sets it to 0). That would be all nice and dandy, but here is the problem, 0 is a valid value because most OS/arches use it, there is no reason 0 is valid (as per POSIX).

And what really gets me is that Pragma... Whoever wrote it probably was getting errors and decided that that was fine. EPERM? EINVAL? Not my problem! No wonder there is a specific s-taprop__linux.adb...

So here we are. NetBSD is not POSIX compliant (min and max SCHED_OTHER priority is -1, which is an error code for the function that should return it), and Linux hardcodes it. Amazing, just amazing...

My solution? Email the NetBSD people. But that won't be enough. So I am thinking of patching the s-taprop__posix.adb file to try it with the default priority, if it fails, with 0, if it fails, with -1 for NetBSD...

Oh well... I thought that the state of GNAT was better... Anyhow, regards,

From: Simon Wright <simon@pushface.org>
Date: Fri, 24 Sep 2021 08:48:34 +0100

> pragma Assert (Result in 0 | EPERM | EINVAL);

EINVAL was added 5 years ago. The others have been there for 20 years (when Ada was added to FSF GCC, according to git blame in https://github.com/gcc-mirror/gcc).

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Fri, 24 Sep 2021 11:44:56 +0200

> EINVAL was added 5 years ago. The others have been there for 20 years

Thank you for your reply Simon. But I think I have understood it now.

It really does not matter what that function "pthread_set..." returns, even if it is an error.

SCHED_OTHER is the default scheduler FIFO and RR are more RTOS-like and are generally reserved for root. I would expect that most programs that spawn threads generally do not care about the priority, since that is managed by the OS.

That would mean that even if that function fails, once the program spawns the actual process, the OS just does it, independently of what the program tried to do. That would explain why it works in OpenBSD, FreeBSD etc, and why I was not getting this error before I added assertions. Because it really does not matter.

I am still very salty about code that knows it fails, but does nothing/is not cleaned up...

I patched however that function and reran ACATS.

Now, I am no longer getting that assertion failure (s-taprop.adb:659). And at the very least the test I worked with (a83a02b) is now fully fixed. However, now, other assertion failures in a couple other places are taking place, primarily s-tassta.adb:1643, which is related to

```
pragma Assert
(Self_ID.Common.Wait_Count = 0);
```

Which, from the comments, the master should not have any slaves but it does somehow (mine is returning a 1). The s-tassta.adb file is shared among all systems (there are no OS specific files). Another common error is s-taprop.adb:463 and STORAGE_ERROR : s-intman.adb:136

I will keep on debugging...

Ada and Other Languages

Ada vs Pascal Efficiency

From: Richard Iswara <haujekchifan@gmail.com>
Subject: Not good for Ada endorsement
Date: Wed, 7 Jul 2021 07:21:55 -0700
Newsgroups: comp.lang.ada

I haven't seen this posted before so apologies if redundant.

A couple days ago this person posted on YouTube this clip: https://www.youtube.com/watch?v=pv4Yq35Chx0.

In the video it was run against Pascal and Ada lost by being 50% slower than Pascal on Prime Sieving. Also shown as 2 orders of magnitude slower than the fastest implementation of that day. See on video at 20:45.

This is the base implementation link: https://github.com/PlummersSoftwareLLC/Primes/blob/

drag-race/PrimeAda/solution_1/src/main.adb.

I know it's trivial, and probably click bait on the video person, but this is not a good PR on Ada reputation. The guy said do a better implementation to get a better score. So can Ada implementation do better?

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 7 Jul 2021 16:06:46 +0100

I watched this finally yesterday as it kept popping up. I started running the entire project last night as running the Ada one on it's own didn't do anything. It's still running, It's been on PrimeR all day!

> So can Ada implementation do better?

Don't know as I don't know the algorithm, but I did clone the repo to look at it. I would be comparing Ada with C, C++, Pascal and any other compiled language. Not just Pascal.

From: Richard Iswara <haujekchifan@gmail.com>
Date: Wed, 7 Jul 2021 20:46:29 -0700

It is supposed to be a basic Sieve of Erastosthenes searching for primes under 1 million.

Odd number search only, can be multithreaded and skip ahead. See the rules at: https://github.com/plummerssoftwarellc/Primes/blob/drag-race/CONTRIBUTING.md#rules.

Indirectly it is a comparison of implementation and tools benchmarking. Looking at the gpr file, there is no compile switch used, not even an "-o2" switch.

From: Jeffrey R. Carter
Date: Thu, 8 Jul 2021 10:20:31 +0200

> Indirectly it is a comparison of implementation and tools benchmarking. Looking at the gpr file, there is no compile switch used, not even an "-o2" switch.

Compiling with -gnatp -O3 would undoubtedly speed it up (suppressing checks is justified since execution with checks active shows that no checks fail).

Looking casually at the code, the map could be replaced by a constant, as Sieve_Size is hard coded to 1,000,000, and the filling of the map is included in the timing. The calculation of the square root of 1,000,000 could be replaced by a constant. The array of Boolean could be constrained to 3 .. Sieve_Size. The function that simply returns (others => True) could be replaced by the aggregate, though optimization will probably do that. Long_Long_Integer could be replaced by a type with range 0 .. 2 ** 31 - 1, though I don't know if that would have any effect. The first inner loop in the sieve algorithm could be eliminated, in which case the initialization of Num could also be removed.

From: Jeffrey R. Carter
Date: Thu, 8 Jul 2021 12:42:25 +0200

Compiling the original code with

gnatmake prime_sieve.adb

gives 408 passes in 5 seconds.

Making the changes listed above (I used Interfaces.Integer_32) and compiling with

gnatmake -O3 prime_sieve.adb

(to ensure that no checks fail) gives 2087 passes in 5 seconds, for a factor of 5.1.

Applying that to the reported 67 passes/second for the original on the test system implies that this version, compiled with checks enabled and optimization, would give 343 passes/second.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 8 Jul 2021 10:42:40 +0200

> Looking casually at the code, [...]

Exactly, this is what is wrong with such contests. The solution is non-scalable giving advantage to low-level languages like C. Scalability and readability has the price that hobby-sized instances work poorly.

P.S. I would also suggest ensuring the Boolean array is not packed. If not with compiler switches and pragmas then by declaring a custom Boolean 1,2,4,8 bytes long depending on the target architecture. It is not a fair play, guys!

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 8 Jul 2021 11:51:03 +0100

Here's the playlist: https://youtube.com/playlist?list=PLF2KJ6Gy3cZ5Er-1eF9fN1Hgw_xkoD9V1

The second video is where he sets up Python, C# and C++.

He shows the C++ jumping from 4645 (https://youtu.be/D3h62rgewZM?t=1246) to 7,436 (https://youtu.be/D3h62rgewZM?t=1306) passes going from 32 to 64 bit.

He also uses clang's -Ofast option to compile for speed over size.

From: Jeffrey R. Carter
Date: Thu, 8 Jul 2021 13:12:29 +0200

Going back to 64-bit integers gives 1999; with -gnatp, 2098

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 8 Jul 2021 18:37:54 +0100

I've uploaded my version here: https://github.com/Lucretia/Primes/tree/add-optimised-ada/PrimeAda/solution_2

It's a straight conversion from the C++ to Ada. I intend to keep optimising it.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 8 Jul 2021 18:43:42 +0100

As a quick test, I removed the Pack attribute from the Bits array and got the following speed:

debug0:

Passes: 1108, Time: 5.003198, Avg: 0.004515521, Limit: 1000000, Count1: 78498, Count2: 78498, Valid: TRUE

Lucretia;1108; 5.003198; algorithm=base,faithful=yes,bits=8

optimised:

Passes: 3286, Time 5.000592, Avg 0.001521786, Limit: 1000000, Count1: 78498, Count2: 78498, Valid: TRUE

Lucretia;3286; 5.000592; algorithm=base,faithful=yes,bits=8

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 09 Jul 2021 01:10:34 -0700

> It's a straight conversion from the C++ to Ada. I intend to keep optimising it.

I'd be interested in seeing a straight conversion of Pascal to Ada, if the existing Ada code differs significantly from the existing Pascal code in a way that affects the speed and isn't a straightforward conversion.

From: Egil H H <ehh.public@gmail.com>
Date: Fri, 9 Jul 2021 01:24:01 -0700

One significant difference between the original Ada version and the Pascal and C++ versions, is that Ada is missing a Num := Factor before the first inner loop.

(Luke fixed that in his version, though)

From: Jeffrey R. Carter
Date: Fri, 9 Jul 2021 11:16:24 +0200

> I get a bit better performance by modifying the check to

>   if Number mod 2 /= 0 and then Sieve.Bits(Index_Type(Number)) then

Since both operands are positive, mod and rem give the same results, and rem is usually a bit faster. It's normally not an issue, but in this case it's done billions of times, so it might make a difference.

From: Jeffrey R. Carter
Date: Fri, 9 Jul 2021 11:21:02 +0200

Note also that short-circuit forms (and then) require disabling processor-level optimizations and may have a negative effect on execution time when used unnecessarily.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Sun, 11 Jul 2021 11:54:27 -0400

>I haven't seen this posted before so apologies if redundant.

It also showed up on the Free-Pascal group. Though there the concern was relative to a C/C++ implementation.

The kicker... Said C/C++ implementation declared practically everything as "constexpr" (or some such), and a Google search indicated that such items are computed... BY THE COMPILER. Not at executable run-time. Run-time basically was "yup, we found them all".

From: Lucretia <laguest9000@googlemail.com>
Date: Thu, 15 Jul 2021 08:13:05 -0700

I managed to get just under 4000 passes with a 1 bit array, but not using Ada's packed array. That's actually the slowest method, strangely.

From: Jeffrey R. Carter
Date: Thu, 15 Jul 2021 17:56:31 +0200

So you have an array of a modular type, calculate an index and bit within it, mask out that bit, and compare it to zero? I would have thought an unpacked array of Boolean would be fastest.

A packed array of Boolean requires all the operations above, plus shifting the bit to the LSB and treating the result as a Boolean, so it may not be that surprising that it's slower.

From: Jeffrey R. Carter
Date: Thu, 15 Jul 2021 23:08:40 +0200

>> A packed array of Boolean requires all the operations above, plus shifting the bit to the LSB and treating the result as a Boolean, so it may not be that

> Don't need to shift to the LSB, only need to shift the 1 to the bit location you want to test, invert and then check against 0.

You know that that is enough, and may be what you're doing manually, but the compiler may not know that. If A is a packed array of Boolean, then A (I) has type Boolean. Unless the compiler can optimize it (and maybe it can), it would normally need to shift the bit down so it can be treated as a value of type Boolean, and then apply whatever you do with the resulting Boolean value.

From: Mace Ayres <mace.ayres@icloud.com>
Date: Sun, 18 Jul 2021 16:03:04 -0700

I doubt if any industrial software engineering in Aviations, railroads, control systems in Europe or in the US, beyond, web programmers are going to abandon Ada, back to Pascal, over such kiddy code comparisons.

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 18 Jul 2021 18:00:19 -0700

It's still a matter of concern if straightforward code is that much harder to compile with Ada, that an advanced Ada compiler (GNAT) produces slower code than a relatively simple Pascal compiler does (depending on what Pascal compiler it was, of course).

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Thu, 15 Jul 2021 09:29:50 -0700

Should fixed lower bound index array (-gnatX) speed it up?

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Fri, 23 Jul 2021 10:04:12 -0700

> Should fixed lower bound index array (-gnatX) speed it up?

Indeed, it does. As posted on Reddit Ada, I got 5476 Passes after changing array types to arrays having fixed lower bound index of 0 on lines 9 and 10 and compiling it using switch -gnatX -02. By the way, I used GNAT Studio 2021 running on Windows 10.

Predefined Integer Sizes in Ada & C

From: Kevin Chadwick <m8il1ists@gmail.com>
Subject: C time_t 2038 problem s-os_lib.ads
Date: Thu, 23 Sep 2021 03:42:16 -0700
Newsgroups: comp.lang.ada

I have noticed that C time_t appears to be Long_integer in GNAT s-os_lib.ads.

Just wondering if it should be 64bit long long as OpenBSD has already moved to long long?

There seemed to be some noise on Twitter about the Linux Kernel side last year but I'm not sure if that ended up just being noise without action or not.

"https://www.openbsd.org/papers/eurobsdcon_2013_time_t/"

p.s. It's interesting that Ada's type system avoids this issue mostly (ignoring leap handling pain)

From: Jeffrey R. Carter
Date: Thu, 23 Sep 2021 16:26:09 +0200

GNAT defines

```
type Long_Integer is range
   -(2 **63) .. +(2 **63 - 1);
for Long_Integer'Size use 64;
```

From: Joakim Strandberg <joakimds@kth.se>
Date: Thu, 23 Sep 2021 08:08:49 -0700

Well, yes Long_Integer is 64-bits, but long long in cpp is 128 bits which sounds like a discrepancy to me. On OpenBSD it indicates C time_t should be changed from Long_Integer to something else that is 128-bits. All packages in Ada have "with Standard; use Standard;" which brings Integer etc. into scope. Long_Integer should be defined in the Standard package. Under help in GPS it should be possible to find the Standard package.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Thu, 23 Sep 2021 12:52:30 -0700

If by "cpp" you mean C++, I've never seen an implementation where long long is 128 bits (or anything other than exactly 64 bits).

In C and C++, int is required to be at least 16 bits (POSIX requires 32), long is at least 32 bits, and long long is at least 64 bits. On most 64-bit Linux-based systems, int is 32 bits, and long and long long are both 64 bits. On 64-bit MS Windows, int and long are both 32 bits, and long long is 64 bits. time_t is 64 bits on almost all 64-bit systems. I've never seen a 128-bit time_t; 64 bits with 1-second resolution is good for several hundred billion years.

If an Ada implementation makes Integer, Long_Integer, and Long_Long_Integer correspond to C and C++'s int, long, and long long, then on a system (e.g.,. Windows) where long is 32 bits, defining time_t as Long_Integer is going to cause problems in 2038 -- *and* it's likely not going to match the system's C and C++ time_t definition.

don't see a definition of "time_t" in s-os_lib.ads on my system.

If an Ada implementation is going to define a type that's intended to match C's time_t, it should match the representation of that C type. I presume GNAT gets this right.

From: Joakim Strandberg <joakimds@kth.se>
Date: Fri, 24 Sep 2021 02:32:54 -0700

Thanks for the summary of different types of integers on different platforms Keith. When I wrote above I had simply done a quick Google search and found https://www.tutorialspoint.com/what-is-long-long-in-c-cplusplus where it said "On Linux environment the long takes 64-bit (8-bytes) of space, and the long long takes 128-bits (16-bytes) of space." I have never seen 128-bit integers either but have seen on the development log on AdaCore's website that support for 128-bit integers has been added to the Interfaces package (Interfaces.Integer_128 and Interfaces.Unsigned_128). I believe they are part of the new Ada2022 standard.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 24 Sep 2021 12:44:38 +0300

Good that they have been added.

> I believe they are part of the new Ada2022 standard.

I believe not. The draft Ada2022 RM still requires no specific integer widths in section B.2, "The Package Interfaces". As in earlier standards, it still says:

"An implementation shall provide the following declarations in the visible part of package Interfaces: - Signed and modular integer types of n bits, if supported by the target architecture, for each n that is at least the size of a storage element and that is a factor of the word size. The names of these types are of the form Integer_n for the signed types, and Unsigned_n for the modular types"

The change by AdaCore probably reflects the fact that gcc now supports 128-bit integers on common platforms.

Wikipedia has a summary: https://en.wikipedia.org/wiki/128-bit_computing.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Fri, 24 Sep 2021 15:54:10 -0700

> Thanks for the summary of different types of integers on different platforms Keith. When I wrote above I had simply done a quick Google search [...]

That web page is simply wrong about long long being 128 bits. It certainly can be (the C standard only says that it's at least 64 bits), but it's exactly 64 bit on every implementation I've seen or heard of.

I'm not shocked that something on tutorialspoint.com is wrong.

There are several common data models in the C and C++ world:

Name	ILP32	LP64	IL32P64
char	8	8	8
short	16	16	16
int	32	32	32
long	32	64	32
long long	64	64	64
pointer	32	64	64

32-bit systems (which are becoming rarer for non-embedded systems) typically use ILP32, and 64-bit Linux/Unix systems typically use LP64. 64-bit Windows uses IL32P64 (and hardly anything else does).

It's *seems* almost obvious that Ada's types

Character
Short_Integer
Integer
Long_Integer
Long_Long_Integer

should correspond to the similarly named C types, but it's not required. (I don't know whether GNAT does so consistently or not.)

Some C and C++ compilers support 128-bit integers on 64-bit systems. gcc supports "__int128" and "unsigned __int128", but they don't quite meet all the C requirements for integer types; for example, there are no literals of those types.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sat, 25 Sep 2021 12:22:17 +0200

> It's *seems* almost obvious that Ada's types [...] should correspond to the similarly named C types

It might turn out as an advantage if Ada programs don't use types named like that.

First, the standard says an implementation MAY provide them.

Second, if Ada programs call C functions that take C int arguments, then argument types taken from Interfaces.C seem to be the obvious choice.

Just state what's needed in the type's definition in your program, referring to "externally defined" types as required.

From: Simon Wright <simon@pushface.org>
Date: Sat, 25 Sep 2021 12:23:53 +0100

> It's *seems* almost obvious that Ada's types [...] should correspond to the similarly named C types, but it's not required. (I don't know whether GNAT does so consistently or not.)

Package Standard in FSF GCC 11.2.0 on macOS (which you can see by compiling something with -gnatS) has

```
type Integer is range -(2 **31) ..
  +(2 **31 - 1);
for Integer'Size use 32;

subtype Natural  is Integer range
  0 .. Integer'Last;
subtype Positive is Integer range
  1 .. Integer'Last;

type Short_Short_Integer is range
  -(2 **7) .. +(2 **7 - 1);
for Short_Short_Integer'Size use 8;
type Short_Integer is range -(2 **15) ..
  +(2 **15 - 1);
for Short_Integer'Size use 16;

type Long_Integer is range -(2 **63) ..
  +(2 **63 - 1);
for Long_Integer'Size use 64;

type Long_Long_Integer is range
  -(2 **63) .. +(2 **63 - 1);
for Long_Long_Integer'Size use 64;

type Long_Long_Long_Integer is range
  -(2 **127) .. +(2 **127 - 1);
for Long_Long_Long_Integer'Size use
  128;
```

I didn't know about the last, which is new in FSF GCC 11/GNAT CE 2021 ... I could build my Analytical Engine simulator with 40 digit wheels (i.e. capable of 40 decimal digits) instead of 50 using Long_Long_Long_Integer instead of GNATColl.GMP.Integer.

Ada Practice

Discarding Function Call Results

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Re: calling function but ignoring results
Date: Wed, 30 Jun 2021 17:06:27 -0700
Newsgroups: comp.lang.ada

> It is not very often that ignoring a function result is okay, but I have run across many instances of the following block structure in code over the years:

```
> declare
>    dont_care : BOOLEAN;
> begin
>    dont_care := foo( x, y );
> end;
```

With, GNAT, this can be:

```
declare
   dont_care : BOOLEAN := foo( x, y );
   pragma Unreferenced (dont_care);
begin
   null;
end;
```

which makes the intent clear. I don't know if Unreferenced was proposed as a language addition; it's not in Ada 202x.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 Jun 2021 22:55:12 -0500

Unreferenced controls warnings (with one exception) are not an Ada concept. So how would we describe what it does? Aspect unreferenced does nothing at all?? :-)

One could imagine an aspect that caused a Legality Rule against an actual reference, but I don't think that is what the GNAT aspect does.

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Date: Thu, 1 Jul 2021 11:07:41 -0700

> > Is there an Ada 202x feature to support calling functions and ignoring the result?

> If you want to use a language that allows this, then you probably shouldn't be developing S/W.

Yes, you are right. But sometimes it is necessary (especially at the H/W level) to force a read of a peripheral register in order to obtain a specific behaviour, e.g., clear an interrupt or latch a value previously written; in these cases what you read is useless.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Fri, 2 Jul 2021 00:32:23 -0700

> But sometimes it is necessary...

Yes, but the frequency is too low to justify yet another feature of the language, IMO.

From: Nasser M. Abbasi <nma@12000.org>
Date: Fri, 2 Jul 2021 20:22:55 -0500

fyi, Matlab had this for years:

https://www.mathworks.com/help/matlab/matlab_prog/ignore-function-outputs.html

"This example shows how to ignore specific outputs from a function using the tilde (~) operator.

To ignore function outputs in any position in the argument list, use the tilde operator. For example, ignore the first output using a tilde.

```
[~,name,ext] = fileparts(helpFile);"
```

From: Matt Borchers <mattborchers@gmail.com>
Date: Fri, 2 Jul 2021 21:59:18 -0700

Thanks for the feedback. I guess I have to live with five lines to accomplish what one should do regardless of the numerous varieties of ways to accomplish this. I mostly appreciate the wordiness of Ada for the clarity it offers to the code maintainers, but in some cases the extra wordiness offers nothing.

Related to this, I really appreciate the new parenthesized expressions as it offers a clean way to declare simple functions. GNAT 21.2 was just released today and includes declare expressions. I don't have the compiler yet, but I wonder if this would work:

```
procedure FOO( x, y ) is
  (declare b : BOOLEAN := foo( x, y ));
```

but it seems likely that an expression returns a value and would not be allowed in this instance.

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 3 Jul 2021 00:37:08 -0700

> I mostly appreciate the wordiness of Ada for the clarity it offers to the code maintainers, but in some cases the extra wordiness offers nothing.

If you use functions properly (only "in" parameters and no side effects) you don't have this issue at all.

Interfacing with C is an edge case which doesn't need to add more noise in the Ada syntax, IMHO.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 3 Jul 2021 10:57:20 +0300

If you use functions properly (only "in" parameters and not side effects) you don't have this issue at all.

However, the problem then changes to ignoring unneeded "out" parameters of procedures, and the only way (currently) is to declare dummy output variables and then leave them unused.

But it is not a big problem, and not worth changing the language, IMO

> Interfacing with C is an edge case which doesn't need to add more noise in the Ada syntax, IMHO.

I agree.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 6 Jul 2021 18:07:38 -0500

> I think any compiler needs some facility like this, at least if it has any pretensions to interfacing to foreign languages

Perhaps "any compiler that tries to warn about unused objects". Janus/Ada doesn't do that, so it doesn't need some facility to turn it off, either. But I grant that if it did have such a warning, then some method to turn it off is needed. (We have a pragma Warning_Level for turning off classes of warnings in selective areas, nothing for individual warnings -- my assumption has been that a warning message might change when the compiler gets upgraded, but the class of warning will not [unless of course there is something else wrong].)

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Fri, 9 Jul 2021 20:14:07 +0200

> In Ada 202x, renaming is easier (assuming the usual case where overloading

> isn't involved):

>   declare

>   Ignore renames Foo (Baz);

>   begin

Is this "type-less" naming a copy of the popular omission schemes like auto in C++? Optional type annotations in Swift, or Scala? Too many of those omissions have invited, uhm, a number of things.

They'll be good, for sure, when securing the workplace semantically; also good for implementers of more complex type inference algorithms and, consequently, for makers of the CPUs that are needed to properly handle the omissions. I think the proper number of omissions is a subject of research at ETH Zürich. They are trying to find a sweet spot that makes inference finish in reasonable time.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 9 Jul 2021 22:20:56 +0300

> Is this "type-less" naming a copy of the popular omission schemes like auto in C++? Optional type annotations in Swift, or Scala?

I don't know all the origins of this language change, but it can be seen as a correction because it avoids the wart in the earlier Ada form of renaming, where a (sub)type name is included. The wart is that the constraints of that (sub)type are essentially ignored, and so can be misleading.

AI12-0275 seems to be the main origin of this change: http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-02751.txt?rev=1.9&raw=N

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 9 Jul 2021 21:57:27 -0500

Right; the name of the type is often a lie vis-a-vis the subtype properties; moreover, it is often the case that the type is included in the renamed item:

```
Foo : renames Some_Type'(Bar(Obj));
```

Repeating the type in such cases is not useful and violates DRY ("Do Not Repeat Yourself"):

```
Foo : Some_Type renames
   Some_Type'(Bar(Obj));
```

We felt this was something that was better handled by style guides rather than imposing unnecessarily wordy syntax.

> AI12-0275 seems to be the main origin of this change

The actual motivation behind this change was a strong desire for a shorter way to write bindings in declare expressions. We tried a number of things, but they all seemed oddly special cases. Eventually, we settled on using normal syntax throughout declare expressions, but simplified the syntax for renaming in all cases. The tipping point was noticing the duplication in examples like the above, along with the fact that the subtype given is ignored anyway.

If we were designing Ada from scratch, the subtype in a rename would have to statically match the nominal subtype of the name being renamed. But that wasn't required in Ada 83 and it would be way too incompatible to require now. (The reason that Ada 83 didn't require it? Jean Ichbiah didn't want to have to define "static matching" -- according to an old thread that John Goodenough dug up. Of course the Ada 9x team decided such a concept was necessary and added it to the language, so we ended up with most constructs using static matching, but renames being different.)

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 12 Jul 2021 08:56:55 -0700

> I think the proper number of omissions is a subject of research at ETH Zürich. They are trying to find a sweet spot that makes inference finish in reasonable time.

I tend to dislike type-inference* almost altogether; I think Ada 2012 and before got it right: very constrained and deterministic contexts (e.g. the for loop's index).

Yes, I realize there's systems like Haskell that are good about types, but as you say these have inferences that take a while. While I'm all in favor of making the compiler do the tedious work, given that types are [in general] a static portion of the program as a whole it should be possible (to borrow from the GPS UI) to throw up the little wrench/auto-fix option and "fill in" the types found so that the next compile doesn't have to infer types... but I suspect that most implementations will instead simply do the inference again and again and again on each compile and waste your time.

* It invites the "could possibly work" C-ish mentality, rather than the "cannot possibly not-work" Ada-mentality, IMO.

Deferred Constants in Bodies

From: Matt Borchers <mattborchers@gmail.com>
Subject: deferred constants
Date: Wed, 7 Jul 2021 12:31:33 -0700
Newsgroups: comp.lang.ada

Is it possible to define a constant in a package body that has its initialization deferred to elaboration?

For example...

```
with Gnat.RegExp;
package body
   pat : constant Gnat.RegExp.RegExp;
begin
   pat := Gnat.RegExp.compile( "..." );
end;
```

Obviously it is not strictly necessary to create 'pat' as a constant, but it is ideal to define symbols as precise as possible. Without it being a constant, the compiler will obviously not check to make sure someone has not inadvertently overwritten it.

GNAT gives me the following errors:

- constant declaration requires initialization expression
- deferred constant is frozen before completion

The first error message is not true, but comes from the fact that the second IS true. Is there a way to postpone the freezing of a symbol until after elaboration?

From: Jeffrey R. Carter
Date: Wed, 7 Jul 2021 22:40:25 +0200

Deferred constants are defined in ARM 7.4 (http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-7-4.html), which says they may only appear in the visible part of a pkg spec, and the full declaration must appear in the private part of the same pkg. So what you're trying is illegal.

In cases like this, you declare the object as a variable, with comments indicating that it is set later and is then constant.

Of course, in the example, one can simply do

```
Pat : constant Gnat.RegExp.RegExp :=
Gnat.RegExp.Compile ( "..." );
```

but there are cases where this is not possible.

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 7 Jul 2021 13:44:32 -0700

Use RENAMES?

```
pat : Gnat.RegExp.RegExp renames
Gnat.RegExp.compile( "..." );
```

Other than this, I would advise not using RegEx.

From: Matt Borchers <mattborchers@gmail.com>
Date: Wed, 7 Jul 2021 16:15:42 -0700

Of course comments help, but the compiler does not enforce what is written in comments. (That I'm aware of.) The fact that the example uses RegExp doesn't matter, it was just code I ran across and the example could be anything. BTW, what's wrong with Gnat.RegExp It has worked in our code for years.

I was looking at old code and began wondering if there was a new or better way to add protection to this entity. It appears not when it is defined in the elaboration block.

I suppose I could just move the call to 'compile' out of the begin block of the package as people have suggested. At what point do constants defined by a function call get elaborated? Before or after the elaboration block? Might I then potentially encounter an elaboration race condition?

The RENAME is interesting as I have not seen that before. Is it a rename of the function call (invokes the function upon reference) or a rename of the function result?

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 7 Jul 2021 16:37:22 -0700

> BTW, what's wrong with Gnat.RegExp?

Regular expressions are for regular languages; it is very easy to violate that restriction with your incoming data.

Most of my professional career has been doing maintenance, and RegEx are *terrible* when it comes to maintainability, to the point that I actively avoid them and advise others to as well, even for things that conceptually *could* be done via RegEx (e.g. recognizing an Integer) in favor of actual parsing... or if you need pattern-matching, something more robust like SNOBOL.

> At what point do constants defined by a function call get elaborated? Before or after the elaboration block? Might I then potentially encounter an elaboration race condition?

This is where the categorization pragmas/aspects come in: if a package that you are depending on are PURE or PREELABORATE then there can be no elaboration error. If the type you are relying on is PREELABORABLE_INITIALIZATION, then there can be no elaboration error. All other conditions are a solid *maybe* on having an elaboration error.

> The RENAME is interesting as I have not seen that before. Is it a rename of the function call (invokes the function upon reference) or a rename of the function result?

That form of RENAMES is the function result.

I've found it an excellent alternative to CONSTANT, as it signals my intent to have an alias for some result inside DECLARE blocks and certain internal objects. (eg Default_Map : Map renames Internal_Map_Generation(P1, P2); ... and then I can use "Default_Map" instead of calling the generation-function at each point and possibly messing things up should the parameters change.)

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 07 Jul 2021 18:21:34 -0700

> Might I then potentially encounter a elaboration race condition?

The Ada rules guarantee no race condition, but sometimes fixing elaboration order gets tricky. GNAT offers (at least) two elaboration models; see the user guide. Normally, you just write the code, and only worry about elaboration if the compiler reports a problem.

Ada and Software Testing

From: Paul Rubin <no.email@nospam.invalid>
Subject: Ada and software testing
Date: Sun, 11 Jul 2021 17:49:56 -0700
Newsgroups: comp.lang.ada

I wonder if there is good guidance around for software testing in the Ada world, or if it depends too closely on the application area. I'm aware of DO-178B and DO-178C in the aviation world, though I haven't studied either of them. Sqlite's document about its testing procedure is also interesting and maybe a cautionary tale. Sqlite is a really nice SQL database whose main misfortune from the Ada perspective is that it is written in C. Its testing doc is here:

https://sqlite.org/testing.html

and a little more info can be found in this interview with the author: https://corecursive.com/066-sqlite-with-richard-hipp/#testing-and-aviation-standards

Overview:

1. Sqlite originally had only ad hoc testing. Then the author (Dr. Richard Hipp) did some work with Rockwell, heard about DO-178B there, and embarked on a large effort to strengthen Sqlite's testing in accordance with DO-178B. Particularly, the Sqlite team created an enormous suite of unit tests aiming to get 100% MC/DC test coverage. That is, for any "if" statement, there must be tests that exercise both branches of the "if". This seemingly got Sqlite to be very reliable.
2. Later on, fuzz testing came into vogue, so they started fuzzing Sqlite. This in fact found a bunch of crashes and vulnerabilities that were duly fixed, and nonstop fuzzing was added to the test setup. But the testing document (section 4.1.6) notes a tension between MC/DC and fuzzing: MC/DC requires deep parts of the code to be reachable by test inputs, while fuzz protection tends to use defensive programming against "impossible" inputs, resulting in seemingly unreachable code. Fuzz testing has been effective enough at finding bugs in C programs that it has now displaced a lot of static analysis in the C world.
3. Sqlite uses a little bit of static analysis (section 11) but the document says it has not helped much. Ada on the other hand uses static analysis extensively, both in its fine grained type system (compared with C's) and using tools like SPARK.
4. Bugs found by fuzz testing C programs are typically the standard C hazards like buffer overflows, undefined behaviour (UB) from bad arithmetic operands, etc. I'm of the impression that Ada is less susceptible to these bugs because of mandatory range checking and less UB in the language.

Well, that went on for longer than I expected. My questions are basically:

Q1. Are there good recommendations for Ada testing strategies Do the tests resemble the stuff in the Sqlite doc?

Q2. Is fuzz testing an important part of Ada testing, and does it tend to find many bugs?

Thanks!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 12 Jul 2021 10:40:24 +0200

> Q1. Are there good recommendations for Ada testing strategies? Do the tests resemble the stuff in the Sqlite doc?

> Q2. Is fuzz testing an important part of Ada testing, and does it tend to find many bugs?

I do not think so.

Here is a war story of a bug I fixed recently. A network protocol implementation used a callback to send the next portion of data, when the transport becomes available.

The callback implementation peeks a portion of data from the outgoing queue and *asynchronously* sends it away. *If* initiation of sending is successful, the queue is popped.

OK?

No, it is a bug that almost never shows itself because initiation of I/O would normally deprive the task of the processor. But if it does not and I/O completes without losing the processor, the callback is called recursively *before* popping the queue and the *same* portion of data is sent again.

Now, neither 100% coverage, nor fuzz, not even 100% black box testing can detect this, arguably trivial bug.

[The fix is to make recursive calls void]

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 14 Jul 2021 12:56:08 -0700

> But if it does not and I/O completes without losing the processor, the callback is called recursively *before* popping the queue and the *same* portion of data is sent again.

This is a garden variety concurrency bug that you're right, wouldn't normally be found with conventional fuzzing, but might be findable with stress testing. A more rigorous approach would involve model checking.

This type of problem happens in C programs all the time as well, and doesn't really signify anything about the effectiveness of fuzz testing. Fuzzing is very effective against C programs, but tentatively maybe less so against Ada programs, because of Ada's more thorough type checking.

> [The fix is to make recursive calls void]

Hopefully there would be some locks between the tasks, though in that case the problem would show up as deadlock.

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Mon, 12 Jul 2021 09:14:39 -0700

> Q2. Is fuzz testing an important part of Ada testing, and does it tend to find many bugs?

You can combine the power of fuzzing with the power of Ada's strong typing, implying standard Ada run-time checks (e.g. range checks), plus a compiler's own checks (e.g. GNAT's validity checks).

Read the following article for details: https://blog.adacore.com/running-american-fuzzy-lop-on-your-ada-code

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 14 Jul 2021 12:32:40 -0700

> Read the following article for details:

Thanks, this is pretty interesting. He runs AFL on three Ada programs: Zip-Ada, and Ada libraries for reading YAML and JSON. It finds bugs in all three, though not very many. It fits my picture that Ada programs are less susceptible than C programs are, to the types of bugs that fuzzing uncovers.

I do have to say that errors thrown by runtime checks on range types are still program bugs, in the sense that they are type errors, that in principle we should want to catch at compile time.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 12 Jul 2021 18:41:28 +0200

> You can combine the power of fuzzing with the power of Ada's strong typing, implying standard Ada run-time checks (e.g. range checks), plus a compiler's own checks (e.g. GNAT's validity checks).

Before the Dark Age of Computing, testing was not arbitrary. You knew things about your implementation and even, God forbid, foresaw some of them.

E.g. if the implementation was "linear" (the case for all buffer overflow stuff) you would simply test the endpoints (extremes) and one point inside instead of wasting time on anything else.

Of course, to make such considerations and techniques work, the programs needed to be designed very differently, which was one of the motivations behind Ada constrained subtypes, ranges etc.

This is also one of the reasons why unbounded strings, dynamic memory allocation etc must be avoided as you leave some upper bounds undefined making a lot of things non-testable.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 14 Jul 2021 21:51:54 +0200

> I do have to say that errors thrown by runtime checks on range types are still program bugs,

No, it depends on the contract.

> in the sense that they are type errors,

A type error cannot happen at run-time per definition of strong typing. Constraint violation is not a type error.

> that in principle we should want to catch at compile time.

If you can. In reality it is impossible to enforce validity per type system, because such contracts are often not enforceable.

So the trick is to relax the contract by including exceptions, which is what Ada constrained subtypes do. But then Constraint_Error becomes a legal "value" function + would "return" on overflow.

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 14 Jul 2021 13:02:23 -0700

> No, it depends on the contract.

If a contract is broken by either the caller or the callee, it is a program bug either way, I would have thought.

> A type error cannot happen at run-time per definition of strong typing. Constraint violation is not a type error.

Hmm ok, if out of range for a range type is considered a constraint error rather than a type error, then it's ok to say the compiler can't check it even in principle, and it becomes the responsibility of the application user or environment. Inputs that trigger a constraint error might be considered invalid in some situations.

> If you can. In reality it is impossible to enforce validity per type system, because such contracts are often not enforceable.

Yep. SPARK tries to enforce such constraints at compile time, but it's not always possible to use it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 15 Jul 2021 09:27:37 +0200

> If a contract is broken by either the caller or the callee, it is a program bug either way, I would have thought.

If the contract includes exceptions, then nothing is broken.

> Hmm ok, if out of range for a range type is considered a constraint error rather than a type error, then it's ok to say the compiler can't check it even in principle [...]

Yes, and the tests must include the cases when exceptions are propagated, which is frequently ignored, though in my view such tests are even more important than the "normal" cases. Exceptions are not likely to happen. So the code not handling contracted exceptions tend to slip into production with catastrophic results.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Fri, 16 Jul 2021 12:01:47 +0200

> My questions are basically:

I'd like to add, if I may, a third question, perhaps a follow-up question, after having been bitten by a bug that was hidden behind assumptions.

Is there a way of systematically looking for hiding places of bugs specifically in places external to the program text? And, then, what kind of mock-ups could establish typical testing patterns? I/O is mentioned in the sqlite examples, but what if you do not assume that there is going to be X equiv. I/O?

Example: Some external library, of very closed source nature, exposes an unforeseen behavior. It turns out that a library function uses a lock, and while waiting for it, the function call times out, the client program reports failure and terminates normally - with side effects...

After the fact, after some reading and then some testing, in an adjusted setup, it all seems plausible. "But, I didn't think of that!". Educated guesses about what the library might do need to be based on a vast set of documents, plus the seller of the library also sells expensive training. Programs need a quick fix, though.

So, what is a proper testing strategy once the programmers have found that the transitive closure of some call might sometimes incur externally caused behavior? Such as timeout, or ordering effect due to concurrency?

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 16 Jul 2021 03:21:24 -0700

> So, what is a proper testing strategy once the programmers have found that the transitive closure of some call might sometimes incur externally caused behavior?

Depending on the situation, this may be an area to try model checking. I've been wanting to try Alloy (alloytools.org) but so far have only clicked around its web site a little. It looks interesting.

From: Paul Butcher <butcher@adacore.com>
Date: Wed, 28 Jul 2021 08:28:55 -0700

If you haven't done already you may also want to have a look at: https://blog.adacore.com/advanced-fuzz-testing-with-aflplusplus-3-00

It's a follow-on blog to the original R&D work around fuzz testing Ada programs and goes into more detail. It also contains an example of why fuzz testing Ada applications over C can actually identify more program anomalies (again by leveraging the power of the Ada runtime checks).

We're actually seeing a lot of interest in fuzz testing Ada programs and a commercial need for an industrial grade fuzz testing solution for Ada.

You may also want to have a look at ED-203A "Airworthiness Security Methods and Considerations" which is a set of guidelines around ED-202A "Airworthiness Security Process Specification". This report explicitly mentions fuzz testing as a means of identifying vulnerabilities and challenging security measures within airborne software.

In addition (and following on from a previous comment) one aspect we are very interested in exploring is being able to bolster existing unit test input data with a fuzzing campaign. Here we would take the existing test inputs and feed them into the fuzzer as the starting corpus (in an automated fashion).

Fuzz testing Ada programs may not currently be a thing, but it soon will be... ;-)

Building GnatStudio 2021 from Sources

From: Rod Kay <rodakay5@gmail.com>
Subject: Building the 2021 source release of GnatStudio
Date: Wed, 28 Jul 2021 19:25:46 +1000
Newsgroups: comp.lang.ada

Has anyone managed this successfully with the Community Edition source release?

As I find it, the source and dependent project sources are out of version sync.

When those problems are sorted out, the python support files installed to '/usr/share/gnatstudio' are problematic at best, to put it 'nicely'. They differ largely from the python support files installed in the corresponding 2021 binary install (and, in fact, break running GnatStudio).

I wonder how this can be (unless I have made several fundamental build errors).

Ada is touted for its safety, stability and portability. What would new-comers think when the main Ada IDE, produced by the main Ada vendor, breaks so frequently (every yearly release, there have been similar difficulties).

How can these problems be 'accidental' over so many years?

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Wed, 28 Jul 2021 03:49:40 -0700

In the same message, you are talking about difficulties with some python files, then mentioning how Ada makes everything bad. Those are two different languages.

I was one of the GPS/GnatStudio developers for quite a number of years (looks like I am still ahead in the total number of commits :-), and a large part of the installation issues (and a somewhat smaller part of the actual stability issues) were largely in the third party libraries that GPS depends on, most notably gtk and friends. Those are very hard to install correctly, they come with tons of dependencies of their own, were not (at the time at least) properly tested on Windows, and so on...

Compiling the Ada part of GnatStudio was not a major issue at the time. I take it that things are more complex now (did not try in 4 years) because there are more dependencies to other Ada libraries. This is a cost to pay for better sharing of code with other projects and the rest of the community (which is something people have been asking a lot). Things could be a lot simpler if gprbuild was a more competent tool similar to what cargo is for Rust for instance. Alire is trying to improve things in that area, so hopefully it will simplify the handling of those dependencies...

Collectively, we certainly owe big thanks to the people out there who build these community packages for others to use. I know Simon does it for macOS, someone else does it on Debian. Not sure whether there is a similar volunteer on Windows.

From: Simon Wright <simon@pushface.org>
Date: Wed, 28 Jul 2021 15:29:16 +0100

> I know Simon does it for macOS, someone else does it on Debian.

Studio is something I've never provided for macOS: up till now, the CE version has been just fine, last one 2019). Pascal (Blady) is working on the 2021 version, I think.

The last time I tried Studio for macOS I ended up in Python hand-managed memory management hell.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 28 Jul 2021 19:49:14 -0500

> Ada is touted for its safety, stability and portability. What would new-comers think when the main Ada IDE, produced by the main Ada vendor, breaks so frequently (every yearly release, there have been similar difficulties).

Since the "main Ada IDE" isn't even an Ada program (primarily being programmed in Python), I'm not sure what it has to do with the reliability of Ada programs. If someone built an all-Ada IDE, then that might make more sense. And in any case, programs like an IDE are almost always installed from binary packages.

From: Roger Mc <rogermcm2@gmail.com>
Date: Wed, 28 Jul 2021 18:09:25 -0700

> Has anyone managed this successfully with the Community Edition source release?

Yes! I once tried to build it and found similar problems. My attempt also included converting many Python 2 sources to Python 3. I note that you refer to "the main Ada IDE" and I tend to agree. The fact that the causes of the problems are due to "python support files ", including version syncing, causing problems with building "the main Ada IDE" and not Ada can quite possibly give a negative impression for Ada even though Ada is not the culprit. I am currently trying to use VS Code, but find the 2019 CE GPS version preferable.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 29 Jul 2021 10:41:29 +0200

First, never ever use Python!

But if you do, as there is no viable scripting alternative (Lua, Julia are only worse and no Ada script fulfills minimal requirements, last time I looked). I do use Python loading it dynamically. That eliminates all build problems, but creates other ones with packaging...

From: Rod Kay <rodakay5@gmail.com>
Date: Thu, 29 Jul 2021 20:29:03 +1000

> Compiling the Ada part of GnatStudio was not a major issue at the time [...]

Building the Ada part of GnatStudio was not the main problem. The difficulty there was only with version mismatches with the Ada dependencies. These were relatively simple to patch by backporting current git code. Though I wonder how these mismatches could exist in the source release when any attempt to build reveals them.

> Collectively, we certainly owe big thanks to the people out there who build these community packages for others to use [...]

I've been maintaining Ada packages for Archlinux for several years now and have had trouble building GnatStudio on each release. Perhaps I was speaking out of accumulated frustration over problems which should be easy to spot and correct (i.e., the dependencies version mismatches).

From: Rod Kay <rodakay5@gmail.com>
Date: Thu, 29 Jul 2021 20:37:55 +1000

> Since the "main Ada IDE" isn't even an Ada program (primarily being programmed in Python), I'm not sure what it has to do with the reliability of Ada programs.

I guess the point I was trying to make was 'Why is GnatStudio using Python at all, given that Ada is superior?'.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 29 Jul 2021 16:37:49 +0200

> I guess the point I was trying to make was 'Why is GnatStudio using Python at all, given that Ada is superior?'.

Python is useful for tailoring; by dropping a Python file in the appropriate directory, you can add any feature to GnatStudio (OK, it needs some XML too).

That's how AdaControl's integration works.

From: Rod Kay <rodakay5@gmail.com>
Date: Thu, 29 Jul 2021 20:47:37 +1000

I ended up installing the files from 'GNAT/2021/share/gnatstudio' to '/usr/share/gnatstudio' which solved most of the Python2/3 problems. There is still an issue with auto-indent, when using the TAB key, which I've not been able to fix.

In case it is of use to anyone, here is a link to the Archlinux gnat-gps package (which builds ok) ...

https://aur.archlinux.org/packages/gnat-gps

Regards.

From: Stéphane Rivière <stef@genesix.org>
Date: Thu, 29 Jul 2021 13:33:52 +0200

> First, never ever use Python!

If scripting capabilities are needed in GnatStudio, why not use HAC? https://github.com/zertovitch/hac

We use it at $job on a daily basis, replacing all our Bash and PHP scripting stuff...

Seven times faster than Bash, tons times more powerful and maintainable and, even better, HAC source can be GNAT compiled from scratch (without changing a line). There is also a shebang to ease scripting like with any other scripting language...

The biggest HAC program here is 3500 lines (!) It's a Cron. A Drupal web scraper to a Wordpress filer and MySQL DB). It syncs every week thousands of product pages and ten thousands of jpeg and pdf files...

A friendly, humble, well tested and capable companion to a first class Ada environment.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 29 Jul 2021 13:58:29 +0200

> If scripting capabilities are needed in GnatStudio, why not use HAC ?

Nothing of the shell sort. I think it is a major confusion on the side of developers of Ada scripts.

For scripting an Ada application one needs support of

1. Loadable modules/packages, prebuilt, to call back to the Ada application

subprograms provided by the module/package;

2. Passing Ada in/out parameters down to a script's subprogram upon invocation;
3. Returning parameters from the script's subprogram;
4. Precompiled script modules, GPS would use a huge number of scripts, compiling them each time would be expensive;
5. Abortable calls and propagation of exceptions out of the script;
6. Concurrent script run-time with independent instances.

For example, this is what I use Python for, and this is just the same case as in GPS:

http://www.dmitry-kazakov.de/ada/max_home_automation.htm#5.1

The user script refers to a preloaded module that offers a communication channel back to the application, e.g. GPS script interface.

In my case the script is called periodically and returns a persistent object, which is passed down by the next call. Such objects must be managed by the caller (the application).

And I load Python dynamically to break dependency on it.

From: Stéphane Rivière <stef@genesix.org>
Date: Fri, 30 Jul 2021 13:29:46 +0200

> For scripting an Ada application one needs support of

I don't see anything that HAC couldn't do, natively or with adaptations, both on the side of HAC and GNATStudio, considering the enormous amount of time that has been spent to integrate Python into GNATStudio.

But, imho, this is all history, GNATStudio is scriptable in Python, GNATStudio is very difficult to build, AdaCore is known to love Python and GNAT CE, as a whole, is a wonderful tool. We have to live with it ;)

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 29 Jul 2021 10:23:46 -0700

> Since the "main Ada IDE" isn't even an Ada program (primarily being programmed in Python), I'm not sure what it has to do with the reliability of Ada programs.

On the issue of IDEs, and in the context of GUI, maybe it would be better to use something like RAPID.

(If it was *just* Windows, I'd recommend Rod look at Claw: pure Ada, no extraneous dependency, and supporting a small vendor.)

At this point, I think it would be prudent (as-a-community) to assess whether or not external dependencies are worth their keep, whether it be a library like gtkAda or GNATCOLL, or whether it be another language like Python. I'm of the opinion that these dependencies hurt Ada's reputation & goodwill (respectively and especially among newcomers and packagers/maintainers) more than they assist Ada's community.

LINKS to Ada-related GUI libraries:

CLA thread on RAPID: https://groups.google.com/g/comp.lang.ada/c/vzajq2ymI0w/m/sOQIfvNRAQAJ

RAPID Website: http://savannah.nongnu.org/projects/rapid/

Paper: https://www.researchgate.net/profile/Martin-Carlisle/publication/221444571_RAPID_A_Free_Portable_GUI_Design_Tool/links/55eeeabe08aedecb68fd812f/RAPID-A-Free-Portable-GUI-Design-Tool

CLAW: http://www.rrsoftware.com/html/prodinf/claw/claw.htm

Paper: http://www.rrsoftware.com/html/prodinf/triadapaper/triada.html

JEWL: http://archive.adaic.com/tools/bindings/JEWL/jewl-17.zip

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 29 Jul 2021 10:43:51 -0700

> Python is useful for tailoring; by dropping a Python file in the appropriate directory, you can add any feature to GnatStudio

I mean we could do that even easier with FORTH, and distribute the Ada implementation alongside it.

If you take a look at the implementation I have https://github.com/OneWingedShark/Forth

-- you'll observe that there's zero non-Ada portions of the program, all the core-words are given in terms of the VM:

https://github.com/OneWingedShark/Forth/blob/master/src/forth-vm-functions.ads

https://github.com/OneWingedShark/Forth/blob/master/src/forth-vm-functions.adb

https://github.com/OneWingedShark/Forth/blob/master/src/forth-vm-default_words.adb

-- and thus we could achieve a completely portable interpreted system.

Having a fully compliant FORTH 2012 interpreter would be pretty nice in this respect, as FORTH is one of 21 ISO programming languages -- https://en.m.wikipedia.org/wiki/Category:Programming_languages_with_an_ISO_standard -- and only one of three which is traditionally interpreted (ECMAScript, ISLISP, FORTH). ISLISP *might* be a better technical choice than FORTH, but the same technique would work in implementing the portability; ECMAScript (aka JavaScript) would be the obvious choice if it were based on popularity.

TL;DR -- There's zero reason to include Python as a dependency in an IDE.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Fri, 30 Jul 2021 04:51:49 -0700

I must admit having a hard time understanding this discussion. There is of course no way AdaCore will change their tools to use any of the suggestions in this thread. HA is 100% unknown outside of the Ada community, and I would guess 100% unknown outside of the small comp.lang.ada subset of it. At least for now, and things could possibly change in the future.

AdaCore has a large number of customers that have written their own integration scripts in python very easily. Those scripts are in general not written by people with knowledge of Ada at all, those are the people responsible for providing the tooling to other teams. So it would make no sense to only have HAC support for instance (and would not remove any of the build difficulties to boot, since backward compatibility is a thing and python would have to be kept)

Shark8 suggested that external dependencies are a bad thing altogether, and libraries like GtkAda and GNATCOLL should never be used. This is totally opposite to what people actually want (see the development of Alire for instance, or what happens in all programming languages out there). So that also makes no sense.

As the original poster mentioned, building GNATStudio is a very difficult thing. Just like building Firefox, or I presume Visual Studio, or any large application out there. Things likely could be improved with better documentation, and that's likely where the community should play a role. AdaCore developers in general have the proper setup because their colleagues helped them (there is no secret documentation that they do not want to publish to the outside), and of course AdaCore cannot test on all systems and all machines out there. But building GNATStudio is something only a few people are intended to do. Others will benefit from their pre-built packages.

From: Shark8
<onewingedshark@gmail.com>
Date: Fri, 30 Jul 2021 09:59:08 -0700

> Shark8 suggested that external dependencies are a bad thing altogether, and libraries like GtkAda and GNATCOLL should never be used.

This is a rather uncharitable take of my suggestion that all dependencies should be, from time to time, evaluated against their benefits and costs.

But I do stand by it: if some dependency costs more to maintain (not excluding things like install/configuration or make/build troubles foisted on the users and maintainers) then it should be eliminated.

I have a rather harsh view of Python itself, especially the tendency to "it works on my computer!" WRT installation woes, that I strongly question if it *IS* "worth its keep".

> This is totally opposite to what people actually want (see the development of Alire for instance, or what happens in all programming languages out there). So that also makes no sense.

I'm sorry, but *HOW* does a package-manager's popularity (much less existence) negate my suggestion that a dependency's usefulness [and pain-points] should be evaluated?

Just because some package's dependency is well-used doesn't make it a good thing to depend upon, does it? I mean, consider the story of NPM and leftpad:

https://www.theregister.com/2016/03/23/npm_left_pad_chaos/

From: Simon Wright
<simon@pushface.org>
Date: Fri, 30 Jul 2021 18:07:55 +0100

[Nothing was quoted. —arm]

Good sense.

> But building GNATStudio is something only a few people are intended to do. Others will benefit from their pre-built packages.

Those of us on macOS have no CE2021 (at least, from AdaCore :-), and CE2020 has no GNATStudio. We don't know what the future holds for other platforms.

Similar position for gnatprove. I don't know how hard it would be to build from the Linux CE2021 sources.

From: Stéphane Rivière
<stef@genesix.org>
Date: Sat, 31 Jul 2021 11:37:02 +0200

> I must admit having a hard time understanding this discussion.

But it's a pleasure to talk with an AdaCore insider :)

> There is of course no way AdaCore will change their tools to use any of the suggestions in this thread.

I didn't even think about it :)

> HAC is 100% unknown outside of the Ada community, and I would guess 100% unknown outside of the small comp.lang.ada subset of it.

It uses a clean subset of a certain computer language we know better than the snake :)

> At least for now, and things could possibly change in the future.

Legacy has to be handled. So I think it's too late and probably irrelevant.

> So it would make no sense to only have HAC support for instance (and would not remove any of the build difficulties to boot, since backward compatibility is a thing and python would have to be kept)

The root of the problem is (to my taste) there. Python has no place in an IDE written in Ada.

I guess GPS/GNATStudio was written also to demonstrate that Ada can implement any complex graphical application.

So, this is a counterproductive example that gives the image of an incomplete or weak language having to use Python to implement a high-level IDE with scripting capabilities.

Emacs uses Lisp, Emacs users script in Lisp (like me at one point).

GNATStudio should have used an Ada subset from the start. The effort was not made. It is too late. Next case :)

> As the original poster mentioned, building GNATStudio is a very difficult thing. Just like building Firefox, or I presume Visual Studio,

That's, to my taste, definitely not a valuable excuse :)

I built GVD (the GNATStudio ancestor) a breeze, almost 20 years ago. Then came GPS, with ton of C inside (nearly 30% due to Berkeley DB embedded at this time) and a "unmakeable" make process :>

> is no secret documentation that they do not want to publish to the outside)

No secret documentation? Okay. So I need it for Linux :) A full GNATStudio build script with instructions, please :) As I don't believe Adacore engineers keep this complex knowledge in their heads...

Anyway, it seems to me that the latest versions of GNATStudio are better finished and the whole thing is a really nice tool to use.

All the best for you and AdaCore team

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 31 Jul 2021 12:30:31 +0200

[...]

> But, imho, this is all history, GNATStudio is scriptable in Python, GNATStudio is very difficult to build, AdaCore is known to love Python and GNAT CE, as a whole, is a wonderful tool. We have to live with it ;)

For AdaCore it is not really much work, they only have to provide a module to interface the GPS engine. Their customers would decide which script they would use.

What AdaCore *must* do is to remove static linking to Python. The GPS user should choose the script language per preferences that would look for the corresponding script run-time e.g. Python or HAC or whatever.

From: Stéphane Rivière
<stef@genesix.org>
Date: Sat, 31 Jul 2021 13:58:37 +0200

> The HAC script must take Argument, call Square (accessible via the module), return the result of Square (Argument).

API HAC has Argument, Argument_Count and Set_Exit_Status, and the result can be piped.

However, I do not state that HAC is production ready for GNATStudio... But HAC is well written and easily hackable (I speak for Gautier ;)

> For AdaCore it is not really much work, they only have to provide a module to interface the GPS engine. Their customers would decide which script they would use.

You're right. That is the best way to handle it. But Emmanuel says that the need for GNATStudio Python is mandatory anyway...

> What AdaCore *must* do is to remove static linking to Python. The GPS user should choose the script language per preferences that would look for the corresponding script run-time e.g. Python or HAC or whatever.

Freedom of choice. I agree. But I guess AdaCore resources are limited and this is like reinventing the wheel.

The free software way could be to fork GNATStudio, simplify it and fully change the build process. Personally, I've neither the time nor the skills to go this way...

The biggest complaint I had about GNATStudio was its instability. I think that AdaCore has made great progress now. It's now a pleasure to work with.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 31 Jul 2021 14:29:04 +0200

> API HAC has Argument, Argument_Count and Set_Exit_Status, and the result can be piped.

Whatever, you could post a complete example, when ready (:-)).

E.g. here is a lesser sample in Julia, an Ada subprogram is called from

Julia back when Julia is called from Ada:

```
with Ada.Text_IO;   use Ada.Text_IO;
with Interfaces.C;  use Interfaces.C;
with Julia;         use Julia;

procedure Ada_Call is
   Bin : constant String := "D:\Julia-1.2.0\bin";
begin
   Load (Bin & "\libjulia.dll");  -- Load library
   Init_With_Image (Bin);    -- Initialize
                             -- environment
   declare
      function Increment (X : Double)
      return Double;
      pragma Convention (C, Increment);

      function Increment (X : Double)
      return Double is
      begin
         return X + 1.0;
      end Increment;
   begin
      Eval_String
        (  "println(ccall("
        &  CCall_Address (Increment'Address)
        &  ",Cdouble,(Cdouble,),10.0))"
        );
   end;
   AtExit_Hook;  -- Finalize environment
end Ada_Call;
```

Note, there is only one process!

> However, I do not state that HAC is production ready for GNATStudio... But HAC is well written and easily hackable (I speak for Gautier ;)

That is not the point. The point is that AFAIK it cannot be used for scripting unless examples as above are provided.

>> What AdaCore *must* do is to remove static linking to Python.

> Freedom choice. I agree. But I guess AdaCore resources are limited and this is like reinventing the wheel.

It is a minimal requirement to replace static linking with dynamic.

Moreover, whatever resources AdaCore has it does not make any sense to call internal GPS functions implemented in Ada from Ada code via Python scripts! So, no work involved.

> The biggest complaint I had about GNATStudio was its instability. I think that AdaCore has made great progress now. It's now a pleasure to work with.

Yes, but each new version of GTK can change that. GTK is unstable on both Windows and Linux, it is just as it is. AdaCore can at best work around GTK bugs.

Though Python is 100% self-inflicted damage. AdaCore could easily implement some Ada script, again, not to confuse with shell. They did it partially with GPR. The GPR compiler could be extended to support a larger variety of expressions.

Customers wanting Python will use Eclipse instead of GPS anyway, so that is not an argument either.

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 2 Aug 2021 18:05:17 -0700

> Yes, but each new version of GTK can change that. GTK is unstable on both Windows and Linux, it is just as it is. AdaCore can at best work around GTK bugs.

And this is exactly why I said that one should evaluate the consequences of your dependencies: depending on something unstable can easily introduce that instability into your program.

> Though Python is 100% self-inflicted damage.

Agreed.

While Python can be quick and "easy" for prototyping, there are whole classes of errors that any dynamic-typed language possesses which statically-typed languages do not. The only dynamically-typed programming language that I've come across which [arguably] has both the mechanisms and culture addressing those issues is LISP.

>AdaCore could easily implement some Ada script, again, not to confuse with shell. They did it partially with GPR.

GPR is a very saddening example. It's too "stringly-typed", it doesn't leverage the obvious structural Ada ancestry, and because of this the GPR-build tool is kneecapped.

(As an example, if GPR files were a tightly restricted GENERIC package [that is, a subset of Ada s.t. all GPR files were valid Ada], with build-options as formal-parameters, you could use the GPR-build tool to automatically generate menus and guide the user through a build.)

[...]

From: Blady <p.p11@orange.fr>
Date: Fri, 27 Aug 2021 11:58:43 +0200

Though GPS mailing list hasn't been used since March 2017, I propose to start a thread about sharing experiences in building GNAT Studio:

https://lists.adacore.com/pipermail/gps-devel/2021-August/000237.html

I've sent a first post with some basic questions about used component versions which are unfortunately not present in INSTALL documentation.

Could please share your experience and component versions on the GPS list? However, it would be nice to get little support from AdaCore staff.

Dynamic Discriminant Problems

From: Simon Wright <simon@pushface.org>
Subject: Discriminant problem
Date: Sun, 29 Aug 2021 19:51:57 +0100
Newsgroups: comp.lang.ada

I have

```
Location_Step.Node_Test :=
  (Node_Test => Get_Node_Type_Test
       (T.Node_Type_Part.all),
   Name  => Null_Unbounded_String);
```

where

```
type Location_Steps is record
   ...
   Node_Test : Node_Test_Specification
     := (Node_Test => No_Node_Test,
         Name => Null_Unbounded_String);
   ...
end record;
```

and

```
type Node_Test_Specification
  (Node_Test : Node_Tests :=
   No_Node_Test) is
   record
      Name : Unbounded_String;
      case Node_Test is
         ...
      end case;
   end record;
```

Because of that function call in

```
Node_Test => Get_Node_Type_Test
   (T.Node_Type_Part.all)
```

the compiler says

value for discriminant "Node_Test" must be static non-static function call (RM 4.9(6,18))

OK, I get that (tiresome though it is, and I'm amazed I've never come across it before), but how to approach it? I seem to be OK with

```
case Get_Node_Type_Test
(T.Node_Type_Part.all) is
   when Text_Node_Test =>
      Location_Step.Node_Test :=
        (Node_Test => Text_Node_Test,
         Name  => Null_Unbounded_String);
   when ...
end case;
```

but this seems very ugly.

I've only just noticed this: I'd been using -gnatX (so that I could use 'Image on records), which meant that the original code was OK (in the sense that it didn't raise any problems), but of course it's not portable even within the GNAT family. I think -gnat2020 might solve the issue too, but there's a bit of a skew between CE 2021 and GCC 11.

From: Simon Wright <simon@pushface.org>
Date: Mon, 30 Aug 2021 09:13:31 +0100

The skew might well be about the semantics of -gnatX - not 100% sure.

The only way I can see that -gnat2020 would help would be if Get_Node_Type_Test was a static expression function[1], but it's not even an expression function! It could be, though since it involves tag tests and can 'return' an exception this seems unlikely.

[1] http://www.ada-auth.org/standards/2xaarm/html/AA-4-9.html#p21.1

From: Jeffrey R. Carter
Date: Tue, 31 Aug 2021 00:03:16 +0200

> value for discriminant "Node_Test" must be static non-static function call (RM 4.9(6,18))

This has always been the rule for aggregates. [...] What I usually do is

```
declare
   Result : Whatever (D =>
   Non_Static_Value);
   -- Discriminant of an object
   -- need not be static
begin
   Result.F1 := ...;
   ...
   Location_Step.Node_Test := Result;
end;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 30 Aug 2021 20:53:41 -0500

> This has always been the rule for aggregates.

But Ada 202x relaxes it slightly. If the expression has a static nominal subtype, and every value in the subtype selects the same variant, then a dynamic discriminant is allowed in an aggregate. I don't know if that is what is happening here or not; this relaxation doesn't come up that often.

From: Simon Wright <simon@pushface.org>
Date: Thu, 09 Sep 2021 20:51:39 +0100

That's exactly what's happening here.

Oddity with Function Returning Image of Fixed Point Type

From: Jesper Quorning <jesper.quorning@gmail.com>
Subject: Oddity with function returning image of fixed point type
Date: Thu, 2 Sep 2021 03:25:45 -0700
Newsgroups: comp.lang.ada

Is something odd going on here? I did not expect Image_Odd1/2 to return floating point images.

```
with Ada.Text_Io;    use Ada.Text_Io;

procedure Fpt_Last is

   type Fpt is delta 0.01 digits 4;
   Image_Last : constant String :=
                     Fpt'Image (Fpt'Last);

   function Image_Ok return String is
   begin
     return Fpt'Last'Image;
     --  return Fpt'Image (Fpt'Last); -- Also ok
   end Image_Ok;

   Last : constant Fpt := Fpt'Last;
   function Image_Odd_1 return String is
    (Fpt'Last'Img);
   function Image_Odd_2 return String is
    (Fpt'Last'Image);
   function Image_Ok_2  return String is
   (Fpt'Image (FPT'Last));
   function Image_Ok_3  return String is
   (Last'Image);
begin
   Put_Line ("Image_Last  : " & Image_Last);
   Put_Line ("Image_Ok    : " & Image_Ok);
   Put_Line ("Image_Odd_1 : " &
                  Image_Odd_1);
   Put_Line ("Image_Odd_2 : " &
                  Image_Odd_2);
   Put_Line ("Image_Ok_2  : " &
                  Image_Ok_2);
   Put_Line ("Image_Ok_3  : " &
                  Image_Ok_3);
end Fpt_Last;
```

```
Output:
Image_Last  :  99.99
Image_Ok    :  99.99
Image_Odd_1 :  9.99900000000000000E+01
Image_Odd_2 :  9.99900000000000000E+01
Image_Ok_2  :  99.99
Image_Ok_3  :  99.99
```

Compiled with gnatmake version 10.3.0 or CE 2020 on macOS 10.13.6.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 02 Sep 2021 10:08:31 -0700

> Is something odd going on here?

Right, this looks like a compiler bug. LRM 3.5(13):
S'Last denotes the upper bound of the range of S. The value of this attribute is of the type of S.

But this is acting like Fpt'Last is universal_real.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 2 Sep 2021 19:32:57 +0200

No, it is this:

LRM 4.10 (40/5)

"X'Image denotes the result of calling function S'Image with Arg being X, where S is the nominal subtype of X."

The question is what is the nominal subtype of Fpt'Last.

From: Jesper Quorning <jesper.quorning@gmail.com>
Date: Thu, 2 Sep 2021 16:24:05 -0700

> No, it is this: LRM 4.10 (40/5)

Had to go back to Ada 83 LRM to find a chapter 4.10.

> "X'Image denotes the result of calling function S'Image with Arg being X, where S is the nominal subtype of X."

LRM 3.5 (35-36) says about the same.

> The question is what is the nominal subtype of Fpt'Last.

Well Image_Ok and Image_Odd_2 should both return Fpt'Last'Image so one of them must be bad.

Found LRM 3.5 (27.7/2) describing the image of a fixed point type.

Thanks for your responses. I will report the issue.

GtkAda Callback and Event

From: Drpi <314@drpi.fr>
Subject: GtkAda callback and event
Date: Sat, 4 Sep 2021 23:39:29 +0200
Newsgroups: comp.lang.ada

I use an event callback with user data.

I first declare a package:

```
package Handler_Motion_Notify is new
  Gtk.Handlers.User_Return_Callback
 (Widget_Type =>
  Gtk.Text_View.Gtk_Text_View_Record,
  Return_Type => Boolean,
  User_Type   => t_Debug_Panel);
```

The function callback is declared like this:

```
function On_Motion_Notify (
   TextView : access Gtk.Text_View.
                 Gtk_Text_View_Record'Class;
   DebugPanel : t_Debug_Panel) return
   Boolean;
```

The connection is done like this :

```
Handler_Motion_Notify.Connect (
   Widget => Panel.TextView,
   Name   => Gtk.Widget.
        Signal_Motion_Notify_Event,
   Cb     => On_Motion_Notify'Access,
   User_Data =>t_Debug_Panel(Panel));
```

This works correctly. But... I need to have access to the event in the callback function. How can I achieve this?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 9 Sep 2021 21:58:03 +0200

> I'm not as versed in GtkAda, but it looks like those have 'Class types so if it is like most of the other GUI frameworks out there, you typically would extend the type that you are doing the handler for and your user data would be fields of the new record type.

> Since the handler uses 'Class you could just cast the parameter to your new type and have access to the user data.

The problem is that GtkAda uses generics instead of tagged types. And, as I frequently say, generics are lousy.

Here is the design, very simplified:

```
generic
   type Widget_Type is new
Glib.Object.GObject_Record with private;--
   type User_Type (<>) is private;
package User_Callback is
   type Int_Handler is access procedure
        (Widget : access Widget_Type'Class;
         Param : GInt;
         User_Data : User_Type);
   procedure Connect
           (  ...,
              Int_Handler
              ...
           );
   type GUInt_Handler is access
   procedure
        (Widget : access Widget_Type'Class;
           Param : GUInt;
           User_Data : User_Type);
   procedure Connect
           (  ...,
              Int_Handler
              ...
           );
   ... -- An so on for each parameter type
```

In reality it is much messier because handlers are created per generic instances. But you see the problem. For each combination of parameters you need a handler type and a connect procedure.

Furthermore, observe that this is inherently type unsafe as you could attach any handler from a generic instance to any event regardless of the parameters of the event.

Welcome to generics, enjoy.

Handlers without user data are non-generic and exist for each event because GtkAda is generated. So, it is possible to generate a non-generic handler/connect pair for each of hundreds of events per widget. This is what Emmanuel suggested:

```
Cb_GObject_Gdk_Event_Motion_
Boolean
```

But, no user data.

You could not do the same and stuff thousands of cases in a single generic handler package! There is only one for all widgets-events.

There is much hatred towards OO design in the Ada community which is possibly a motive behind this.

An OO design would be to create an abstract base type for each event with a primitive operation to handle the event. The target widget type would be fixed class-wide, but since it is practically never used, that is no problem.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 9 Sep 2021 23:56:21 -0700

>> type My_Button is new Gtk.Whatever_Path.Button_Type with record

>>   User_Data : User_Data_Type;

>> end record;

This is indeed the recommended approach. In practice, most widgets have a single callback per event type (so one for motion_notify, one for click, and so on). All that's needed is the `Self` parameter which contains all relevant information. At least this was my experience based on the GPS source code, which is a relatively extensive GUI application. I don't remember the stats exactly, but there were just a few cases where this approach did not work. And this is why we implemented the higher-level approach in GtkAda, where there are no possible errors in the type of parameters for callbacks.

There are a few cases where you want to share the same callback subprogram for multiple events, or multiple types of widgets for instance. In these cases, you might have to fallback to using the generics to connect, along with specifying a user data. As much as possible, I would recommend not using this approach if you can avoid it.

I do not share Dmitry's distrust of generics, but for GUI applications I 100% agree that an OO approach works much better indeed. The performance cost is negligible in such contexts, and the flexibility is much needed.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 10 Sep 2021 22:58:06 +0200

> I didn't think I could extend the widget type to add my own parameters.

Not only for parameters.

Even more frequent purpose is a composite widget. E.g. consider a text edit widget with a scroll bar, a menu, some buttons etc. Typically, you would take some existing widget and derive your widget from there.

In the Initialize you will create all other widgets and pack them into the widget (if it is a container) or Ref them otherwise. In Gtk_New you will have custom parameters.

This new widget you could use just as any built-in widget. Moreover, you can create a new class for your derived widget and add new events, properties and styles for external parametrization etc. The styles can be set via CSS.

It is a very powerful and versatile mechanism GtkAda offers.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sat, 11 Sep 2021 00:38:07 -0700

> Typically, you would take some existing widget and derive your widget from there.

I agree this is exactly the right design and the way we intended GtkAda to be used (supporting this was not completely trivial at first, though later versions of gtk+ made that simpler).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Sep 2021 17:56:02 +0200

> Any pointers on how to use CSS styles?

This is GTK documentation of CSS: https://docs.gtk.org/gtk3/css-overview.html

Here is an example of a custom button that uses CSS for label, icon, tool tip etc.

http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm#Gtk.Generic_Style_Button

From: Adamagica <christ-usch.grein@t-online.de>
Date: Sun, 12 Sep 2021 00:08:59 -0700

> The big problem is documentation.

A good introduction into GtkAda is direly needed. Trial and error cost me enormously much time.

The GtkAda UG and RM are a bad joke.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 12 Sep 2021 10:52:34 +0200

> The GtkAda UG and RM are a bad joke.

I think you rather mean GTK introduction because GtkAda follows GTK to the letter. There are few advanced topics of interplay between GtkAda objects and GObject etc, but that is not required in the beginning. Basically, you know GTK, you know GtkAda.

Regarding GTK introduction, it would require a genius to write that. GTK is incredibly messy and full of small details you must know before you start. I have no idea how anybody could summarize that in a compact form. There exist various GTK "getting started." All of them, while describing important things, miss minor details essential to write an actual application. There seems to be no such thing as an overview in the case of GTK.

And things are far worse for those who get lured by GLADE. GLADE further obscures what is going on, what has to be done. Be happy you did not step into that...

From: Drpi <314@drpi.fr>
Date: Sun, 12 Sep 2021 15:00:52 +0200

> I think you rather mean GTK introduction because GtkAda follows GTK to the letter.

I don't fully agree with you. I already found Gtk examples I've not been able to use directly with GtkAda because of Ada implementation. Or at least, it was not the best way to do things. Mostly due to the GtkAda OO implementation. Events and customized widgets are good examples.

I also lose a big amount of time searching for how to do things. [...]

I think the tutorial I've found is a good one to start. It starts from scratch which is not as easy as one could think it is: a basic application never ends. The main window closes but the exe never stops. Quite disturbing. When you create a basic GtkAda project with GPS, you get an application with such a behavior. At first, I thought I did something wrong when installing GtkAda. I then found the tutorial and discovered this behavior is the correct one.

> And things are far worse for those who get lured by GLADE.

I tried Glade once and quickly changed my mind. I don't say it's a bad tool. Just that, like you said, things are more obscure using it. As I like to understand what I do, this is not the way to go for me right now. I did the same thing with WxPython. I learned to construct my GUI programmatically. Then, when I've been comfortable with it I switched to wxFormBuilder for some of my projects.

URL to the French tutorial I use:

https://zestedesavoir.com/tutoriels/645/apprenez-a-programmer-avec-ada/555_ada-et-gtk-la-programmation-evenementielle/2676_gtkada-introduction-et-installation/

One problem with this tutorial is that it is outdated.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 12 Sep 2021 15:57:55 +0200

> I don't fully agree with you.

> I already found Gtk examples I've not been able to use directly with GtkAda because of Ada implementation. Or at least, it was not the best way to do things.

Well, you are supposed to use GValues in GTK. GtkAda just makes life much easier for you by adding a typed layer to connect and handle the events. How it works is well explained in the documentation. E.g.

https://docs.adacore.com/live/wave/gtkada/html/gtkada_ug/signals.html

The problem is that you started with that, skipped reading for later because it was too much reading. If you had started the GTK's way first, namely with GValues, then after pulling much hairs, read the stuff more carefully, then you would rather say, "Aha, that is a much better way to do this. Thanks."

> Mostly due to the GtkAda OO implementation. Events and customized widgets are good examples.

Right, because when you begin with GtkAda with no prior knowledge of GTK, you are at a complete loss.

Luckily, you do not yet understand how deep the abyss is! (:-))

> I also lose a big amount of time searching for how to do things.

Because GTK is a mess. Nobody ever knows how to do this or that in GTK. I keep the GTK sources at hand to consult to just understand what is going on. [It was a lot easier a decade ago, when Google was a search engine and GTK topics were not spammed into oblivion by Python and C# garbage sites.]

> I think the tutorial I've found is a good one to start. It starts from scratch which is not as easy as one could think it is: a basic application never ends. The main window closes but the exe never stops. Quite disturbing.

Then the tutorial is broken. If you look at GtkAda samples and tests they all contain the basic GTK frame with On_Delete_Event and On_Destroy.

Trivial Question: How to Avoid Confusing Sec, Min, Hour and Day in a Program?

From: Reinert <reinkor@gmail.com>
Subject: Trivial question: how to avoid confusing sec, min, hour and day in a program?
Date: Sat, 4 Sep 2021 23:56:43 -0700
Newsgroups: comp.lang.ada

Anybody with good ideas on how (in a simplest possible way) to avoid to confuse between representation of time as seconds, minutes, hours and days in an Ada program? Standardize on internal representation of time as seconds (Float)? I will delay to use complex approaches for physical units.

It is somewhere in my program natural/human to think in seconds whereas minutes or hours feels more natural at other places (so the numerics is "human"). Example to illustrate: heart rate is "natural" to give in number per minute (not in number per second, hour or day). Time on work is normally given by hours (not seconds) etc.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 5 Sep 2021 09:27:38 +0200

> Anybody with good ideas on how (in a simplest possible way) to avoid to confuse between representation of time as seconds, minutes, hours and days in an Ada program?

Just use the standard type Duration.

> It is somewhere in my program natural/human to think in seconds whereas minutes or hours feels more natural at other places (so the numerics is "human"). Example to illustrate: heart rate is "natural" to give in number per minute (not in number per second, hour or day). Time on work is normally given by hours (not seconds) etc.

The user interface is responsible for converting anything to Duration and back.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sun, 5 Sep 2021 15:42:49 +0300

> Just use the standard type Duration.

But note that:

- Duration is a fixed-point type, not floating point, so it has a fixed resolution, Duration'Small.
- The resolution and range of Duration are implementation-defined, and the standard does not guarantee much. RM 9.6(27) says:

"The implementation of the type Duration shall allow representation of time intervals (both positive and negative) up to at least 86400 seconds (one day); Duration'Small shall not be greater than twenty milliseconds."

(Aside: I wonder why this paragraph is not in RM A.1, "The Package Standard", where the Duration type is introduced.)

If you want your code to be portable, define your own type for "time in seconds", choosing the properties your application needs.

That said, I believe that GNAT versions typically provide a 64-bit Duration type that has enough precision and range for most applications dealing with times on human scales. But perhaps not on nuclear, geological or astrophysical scales.

From: Ldries46 <bertus.dries@planet.nl>
Date: Mon, 6 Sep 2021 09:20:27 +0200

I agree with Dimitry but I think the problem is far bigger and not only in Ada but in every computer language. You cannot use packages written by others if you do not know the dimensions they use. For instance in mechanical engineering in Europe there is a general use of cm while in other countries they use inches and in aircraft engineering they use in Europe mostly mm.

In general the use of dimensioning systems (f.i. cgs -cm gram second) is different for different countries and application areas. Some factors used within calculations are even dependent on the dimensioning system. It would be a good idea if a package was developed that can solve all these problems for instance by doing the calculations in one of the possible dimensioning systems automagically presenting the results in the dimension you choose.

Such a system must exist of a record for every value, that exists of at least real values for the standard dimensioning system and the value in the dimensioning system you want, the factor between the two systems and the string containing the dimension, perhaps even more. There should be functions for "+", "-", "*", "/" and sqrt. The calculation should be done for the value in the standard dimension while the input and output must be in the wanted system. Also the strings and factors must be changed when necessary.

I have already been thinking how but the problems are mostly in the strings and the fact that there are also dimensions that have a lower limit (temperature) or have an offset (degrees Celsius, Reamur or Fahrenheit) or exist of several integer values (time). I think that the potential of Ada of being independent of an operating system can be extended that way to independent of the dimensioning system.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 6 Sep 2021 11:47:52 +0200

> I agree with Dimitry but I think the problem is far bigger and not only in Ada but in every computer language.

Why do you need that? If written in Ada, the value is of some separate numeric type you could not mix with other types.

> In general the use of dimensioning systems (f.i. cgs -cm gram second) is different for different countries and application areas.

If calculations are involved, they are performed in SI, because otherwise you need factors in all formulae. And SI means no mm, but m, no km/h, but m/s etc.

> There should be functions for "+", "-", "*", "/" and sqrt.

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

From: Ldries46 <bertus.dries@planet.nl>
Date: Mon, 6 Sep 2021 15:06:17 +0200

> [...] If calculations are involved, they are performed in SI, because otherwise you need factors in all formulae.

Sorry but It should perhaps be so but it is not. Programs made primarily for Aircraft engineering (f.i. CATIA 5/6) do use mm and your velocity clock in your car shows km/h and it registers the distance in km (in England and the USA mph and miles) In aircraft the Speed is often still measured in Knots (Nautical miles per hour). Maybe French aircraft will possibly show km/h. It is too hazardous to change this.

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 6 Sep 2021 15:43:45 +0200

> Sorry but It should perhaps be so but it is not.

Do not confuse computations with input/output. I agree with Dmitry, all computations should be performed in SI, with a (user selectable) choice of units for input and display.

From: Ldries46 <bertus.dries@planet.nl>
Date: Mon, 6 Sep 2021 16:13:36 +0200

> Do not confuse computations with input/output.

What I tried to say is just what J-P Rosen says. But in my experience (41 years in aircraft engineering) that is not so, every time you have to realise what the system you're working with and you are forced to even use systems parallel. One of the points I made is in mechanical engineering which often uses cm has standard profiles using mm HE110B is 100mm high. That is the reason that I ask for a package which solves this problem for once and always and still gives the user the possibility to use all kind of dimensions

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Mon, 06 Sep 2021 11:10:49 -0400

>100mm high. That is the reason that I ask for a package which solves this problem for once and always and still gives the user the possibility to use all kind of dimensions

So... a reimplementation of the HP-48 UNITS module... Which requires all values to have a unit designation attached (eg: 1.6_km) and internally probably tracks the input unit but converts to base (SI) units for computations, then remaps to user input units for display.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Mon, 6 Sep 2021 08:55:57 -0700

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

http://archive.adaic.com/tools/CKWG/Dimension/Dimension.html

You might try to improve one of those packages for your needs.

Don't know what HE110B is.

From: Reinert <reinkor@gmail.com>
Date: Sat, 23 Oct 2021 23:52:17 -0700

Ada seems to guarantee that Duration covers 24 hours (?). What do you do when you need to represent for example 5 years?

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 24 Oct 2021 09:24:46 +0200

> Ada seems to guarantee that Duration covers 24 hours

Ada /guarantees/ at least 24 hours, since subtype Day_Duration corresponds to one day.

In practice, Duration is much bigger. There is no requirement for it, since it obviously depends on the implementation. With GNAT, the following program:

```
with Text_Io; use Text_Io;
procedure Test_Duration is
   package Duration_Io is new Fixed_Io (
      (Duration);
   use Duration_Io;
begin
   null;
   Put ("duration'last:");
   Put (Duration'Last); New_Line;
   Put ("days:");
   Put (Duration'Last / 86400); New_Line;
   Put ("years:");
   Put (Duration'Last / 86400 / 365.25);
   New_Line;
end;
```

gives:

```
Duration'last:   9223372036.854775807
Days:        106751.991167300
Years:         292.271023045
```

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 24 Oct 2021 12:39:34 +0200

>   Put ("years:") ; Put (Duration'Last / 86400 / 365.25); New_Line;

Still using the Julian calendar? In the Common calendar, the average length of a year is 365.2425 days (97 leap years every 400 years).

> years: 292.271023045

Should be 292.277024627

From: Simon Wright <simon@pushface.org>
Date: Sun, 24 Oct 2021 11:48:24 +0100

> What do you do when you need to represent for example 5 years?

This must depend on your use case.

I'd imagine you want to arrange for some event to happen 5 years in the future. The 'natural' way to do this might be, in a task,

```
delay until Ada.Calendar.Clock +
   Duration'({5 years});
```

but this comes up against two problems: first, as you note, that long a duration might not work, and second, the computer is almost certain to have been restarted by then, losing this task.

The second problem could be solved by, e.g., keeping a backed-up time-ordered queue of events to be processed, with a task that delays until the next event is due.

As for the first -- I think you may need to make an appropriate Duration'Last part of your compiler selection criteria.

Single-Instance Executable, TSR-style Programs, "lockfiles" and the DSA

From: Shark8 <onewingedshark@gmail.com>
Subject: Single-Instance Executable, TSR-style programs, "lockfiles" and the DSA
Date: Wed, 8 Sep 2021 14:23:58 -0700
Newsgroups: comp.lang.ada

I'm currently engaged in writing a series of programs for some scientists to control a few cameras; one system is a sort of cobbled together web-program, distributed across several computers [PHP for interface + C++ for message-slinging and camera-control], while the other is a single computer basically running the camera's manufacturer's program. -- This is mostly about the latter, though the former will need to be addressed [via DSA(?)] in the near future.

There is another system that I'm not touching (for now) which uses lockfiles; sometimes (crashes and erroneous shutdowns) will leave the lockfiles behind. I have a controlled type-wrapper that will close its file if it is still open, and that could easily be adapted to delete them on finalization in the case of a lockfile. -- (#1) What is the best way that the community has come up with regarding lockfiles or similar functionality?

In this particular case, the lockfile represents that the control software for a particular instrument is already running, which brought to mind the old DOS TSRs where you would boot up the program and could call it (or another program using its services) again to achieve some different/special effects, which then brought to mind the new single-instance executables. Now, obviously the DSA can be used in this manner so that one partition provides services and the client partition queries/quits as needed. -- (#2) Is there a non-DSA, and hopefully portable, Ada way to achieve single-instance executables? [I haven't had any luck trying web-searches on this topic.]

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Fri, 10 Sep 2021 00:10:49 -0700

When I wrote the program that processes all incoming email on the mailing lists at AdaCore (in particular to manage tickets), we were using lockfiles indeed to coordinate between all the instances of the program (one per incoming email). The lockfile contained an expiration date and the PID of the process that took the lock, and a program was allowed to break the lock when that date was reached (like 10min or something, I forgot the value we came up with, when processing one message takes a few milliseconds), or when the process no longer existed (so crashed). So at least the system could not totally break and would eventually recover.

This is of course far from perfect, since during those 10 minutes no email could be processed or delivered, and if the timeout is incorrect we could end up with two programs executing concurrently (in practice, this was not a major issue for us and we could deal with the once-a-year duplicate ticket generated).

Years later, we finally moved to an actual database (postgresql) and we were able to remove the locks altogether by taking advantage of transactions there. This is of course a much better approach.

When I look at systems like Kafka (multi-node exchange of messages), they have an external program (ZooKeeper) in charge of monitoring the various instances. Presumably a similar approach could be used, where the external program is much simpler and only in charge of synchronizing things. Being simpler and fully written in Ada, it would be simpler to ensure this one doesn't crash (famous last words...).

From: Shark8 <onewingedshark@gmail.com>
Date: Fri, 10 Sep 2021 09:26:08 -0700

> Years later, we finally moved to an actual database (postgresql) and we were able to remove the locks altogether by taking advantage of transactions there.

Interesting.

When you moved to DB, did you use the DSA to have a Database-interface partition and client-query/-interface partition? I'm assuming not, because such a ticketing system probably doesn't have enough need for distributed clients, report-generators, etc. to justify such a design.

> When I look at systems like Kafka (multi-node exchange of messages), they have an external program (ZooKeeper) in charge of monitoring the various instances.

I suspect such designs are consequences of the poor support for processes/tasking that C has; the Ada equivalent of the functionality would be to have a TASK dedicated to the DB-interfacing [assuming single-node]; for full multi-node DB-backed/-transacted message-exchange DSA makes a lot of sense: Partition your DB-interface into a single node, then have your clients remote-interface that node.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sat, 11 Sep 2021 00:42:19 -0700

> When you moved to DB, did you use the DSA to have a Database-interface partition and client-query/-interface partition?

We did not use the DSA. Postgres itself is a very capable server, which is implemented way more efficiently (and tested way better) than we could ever do I think, since this was only a side job. No reason to add an extra layer between the mail-processing program and the database.

> I suspect such designs are consequences of the poor support for processes/tasking that C has; the Ada equivalent of the functionality would be to have a TASK dedicated to the DB-interfacing.

I don't think you need a task dedicated to the database. Postgres handles concurrency very efficiently, it can do asynchronous queries if you really need that, and so on. If you indeed have a database in your application, you could also use that to handle inter-process locking (pg_advisory_lock() for instance)

GtkAda and €

From: Adamagica <christ-usch.grein@t-online.de>
Subject: GtkAda and €
Date: Fri, 10 Sep 2021 10:56:19 -0700
Newsgroups: comp.lang.ada

I'm struggling to get the euro sign in a label or on a button in GtkAda. I have the euro sign on my German keyboard (on the E key), but I have no idea how this is encoded. So how do I get this in UTF8?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 10 Sep 2021 20:53:07 +0200

> So how do I get this in UTF8?

With Strings Edit:

```
Strings_Edit.UTF8.Image (16#20A0#)
```

See:

http://www.dmitry-kazakov.de/ada/strings_edit.htm#7

Otherwise, see:

https://www.fileformat.info/info/unicode/char/20ac/index.htm

It gives the hexadecimal UTF-8 encoding:

0xE2 0x82 0xAC

So, in Ada:

```
Character'Val (16#E2#) &
Character'Val (16#82#) &
Character'Val (16#AC#)
```

From: Adamagica <christ-usch.grein@t-online.de>
Date: Sat, 11 Sep 2021 02:20:32 -0700

Is there no way to use the character € directly? Imagine, you want to write cyrillc on the label of a GUI? Would you use hex values or would you write “Я говорю по-русски”.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 11 Sep 2021 12:04:40 +0200

I would never use Cyrillic in the source code.

Anyway, this is a question regarding the encoding of literals in Ada. Ada 2X supports Unicode and GNAT supports Unicode sources.

I never tried it but I suppose the following should work:

```
Strings_Edit.UTF8.Handling.To_UTF8 ('€');
```

Here '€' should be resolved to Wide_Character'('€') and then converted to a UTF-8 encoded String.

As for labels, icons etc, I use GTK style properties.

I.e. Let me have a label with a text on it. Usually this label would be packed in some larger container widget, e.g. Gtk_Grid_Record. I derive a custom widget from Gtk_Grid_Record. Then I call Initialize_Class_Record once to create the new widget "type" (used G_New). There I add style properties like this:

```
   Class_Record : aliased
Ada_GObject_Class := Uninitialized_Class;
   ...
   procedure Initialize
           (Widget : not null access
             My_Widget_Record'Class
           ) is
   begin
     G_New (Widget, Get_Type);
     -- Get_Type will register class
     ...

   function Get_Type return GType is
   begin
     if Initialize_Class_Record
        (Ancestor     => Gtk.Grid.Get_Type,
           -- Parent class
         Class_Record =>
              Class_Record'Access,
          Type_Name    => "mywidget"
        )  then -- Not yet registered
        Install_Style_Property
        (  GLib.Types.Class_Ref
          (Class_Record.The_Type),
          Gnew_String
          (  Name     => "label",
             Nick     => "Label",
             Blurb    => "Label text I want to be
                           able to change",
             Default => "I speak English"
        )  );
        ...
     end if;
     return Class_Record.The_Type;
  end Get_Type;
```

The widget must handle "style-updated" from where it would use Style_Get to get the label text and set it into the label.

So, a Russian localization would be a CSS sheet file defining the property "label":

```
--8<--
mywidget {
   -mywidget-label: "Я говорю по-русски";
}
--8<--
```

P.S. Inventors of GTK CSS sheets apparently misspelled the word "sheet", they should have used the letter 'i'! (:-))

From: Emmanuel Briot
<briot.emmanuel@gmail.com>
Date: Sat, 11 Sep 2021 04:11:35 -0700

> custom widget from Gtk_Grid_Record. Then I call Initialize_Class_Record once to create the new widget "type" (used G_New). There I add style properties like this:

I am impressed! I have never had the courage to actually use those properties in my code...

I would use GtkAda.Intl, so that the code would contain

```
use GtkAda.Intl;
Button.Set_Label (-"string to translate");
```

and the translations are given in a separate file.

This is also theoretical for me: although we had initially tried to maintain such a translation file for GPS (and made sure that all user-visible strings on the string were used with the "-" operator in case we ever wanted to do a translation, it was never done in practice).

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 11 Sep 2021 14:47:53 +0200

> I am impressed! I have never had the courage to actually use those properties in my code...

I used properties because I had custom general-purpose widgets rather than an end application like GPS.

A nice thing about GtkAda is that one can use all GTK stuff without any C insertions and it is highly extensible.

From: Adamagica
<christ-usch.grein@t-online.de>
Date: Sat, 11 Sep 2021 06:26:29 -0700

> Here '€' should be resolved to Wide_Character'('€') and then converted to a UTF-8 encoded String.

This does not work. Source files are in Latin_1 by default and € is beyond 255, so GNAT cannot handle '€'. I tried to set the source file's character set to Unicode UTF16 (in GPS, from the file's context menu choose "Properties...") with terrible effects. A real no-go.

> As for labels, icons etc, I use GTK style properties.

I dare not try this...

From: Adamagica
<christ-usch.grein@t-online.de>
Date: Sat, 11 Sep 2021 06:51:40 -0700

Being German, I need umlauts and € together in strings to write them to some labels. Using Character'Val (16#E2#) & Character'Val (16#82#) & Character'Val (16#AC#) complicates things, since umlauts are above 255 and need transformation to UTF8, whereas the euro sequence above is already in UTF8 and must not again be transformed.

What a mess!

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 11 Sep 2021 16:13:07 +0200

> What a mess!

Huh, the mess here is Latin-1 introduced by Ada 95, no such thing should have been even supported. This happened because in the 90s UTF-8 was not yet established, so Ada 95 made Character Latin-1 and added Wide_Character for UCS-2. This was a huge mistake with wide (pun intended) reaching nasty consequences.

Since the Ada type system is too weak to handle encodings, Strings should simply be UTF-8 and Character an octet with lower 7-bits corresponding to ASCII.

Anyway, for anything that is not ASCII I use a named constant.

From: Manuel Gomez
<mgrojo@gmail.com>
Date: Sat, 11 Sep 2021 19:46:46 +0200

> Being German, I need umlauts and € together in strings to write them to some labels.

When converting to UTF8, can you specify that you are using Latin-9 (ISO-8859-15), instead of Latin-1? Latin-9 is equivalent to Latin-1 plus the Euro sign, instead of the generic currency sign, since Latin-1 predates the Euro.

In that case, it would be:

```
Euro_Sign : constant Character :=
                Character'Val (164);
```

This is from Ada.Characters.Latin_9, provided by GNAT (not in the standard). Not sure, but maybe you could type the Euro sign in the source code with the keyboard, since the representation is the same.

Another option is to use ASCII only (with some encoding for umlauts and Euro sign) and then apply localization for the strings that must be "translated" to proper German.

From: Adamagica
<christ-usch.grein@t-online.de>
Date: Sun, 12 Sep 2021 00:04:47 -0700

> This is from Ada.Characters.Latin_9, provided by GNAT (not in the standard).

Not sure, but maybe you could type the Euro sign in the source code with the keyboard, since the representation is the same.

In GNAT Studio, you can set the encoding (from the file's context menu choose "Properties...") to Latin_9. Then the character 164 is displayed as € in the Ada source file. You can even use the € key on the keyboard. That does not help, however, since Unicode is based on Latin_1, and when this is transformed to UTF8, the currency character appears on the GtkAda GUI.

> Another option is to use ASCII only (with some encoding for umlauts and Euro sign) and then apply localization for the strings that must be "translated" to proper German.

I indeed use Character 164 as a placeholder in the Ada source code. When transforming to UTF8, I search for this character first, transform the head string, insert the Euro sequence and transform the tail string recursively. This works.

From: Manuel Gomez <mgrojo@gmail.com>
Date: Sun, 12 Sep 2021 13:44:54 +0200

> In GNAT Studio, you can set the encoding (from the file's context menu choose "Properties...") to Latin_9. Then the character 164 is displayed as € in the Ada source file. That does not help, however, since Unicode is based on Latin_1, and when this is transformed to UTF8, the currency character appears on the GtkAda GUI.

I suppose this is because the conversion assumes Latin-1 input, and it is acceptable given that String type should be in that encoding, but with a general string conversion library, like iconv, you can convert between any 8-bit character encoding and UTF-8.

Here an Ada binding to iconv (I haven't used it): https://github.com/ytomino/iconv-ada

And Matreshka League has several 8-bit character encodings, although it lacks ISO-8859-15. It should be easy to add ISO-8859-15 based on ISO-8859-1: http://forge.ada-ru.org/matreshka/wiki/League/TextCodec

But I guess what you are already doing is the easiest approach. It's just one character.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Mon, 13 Sep 2021 00:21:05 -0700

> What a mess!

Character encoding in source code, input-output and GUI is a known mess. There is Ada 2022 library that provides high level API to process text information, see

https://github.com/AdaCore/VSS

You can try to use Virtual_String everywhere, and do encoding conversion only to get/pass text from/to Gtk+ or input-output streams.

Note, if you want to write characters outside of the ASCII range in the source code you will need to use UTF8 for source files and provide -gnatW8 switch to compiler. It may break compilation of old code sometimes :(