

The journal for the international  
Ada community

# Ada User Journal



Volume 42  
Numbers 3-4  
Sep-Dec 2021

<b>Editorial</b>	129
<b>Quarterly News Digest</b>	130
<b>Conference Calendar</b>	170
<b>Forthcoming Events</b>	179
<b>Articles from the AEiC 2021 WiP Session</b>	
K. Nyborg Gregertsen <i>Ember: an Embedded Robotics Library in SPARK</i>	185
D. García Villaescusa, M. Aldea Rivas, M. González Harbour <i>Queuing Ports for Mesh Based Many-Core Processors</i>	189
<b>AEiC 2021 Industrial Presentations</b>	
A. Marriott, U. Maurer <i>More Ada in Non-Ada Systems</i>	193
M. Martignano <i>Static Analysis for Ada, C/C++ and Python:     Different Languages, Different Needs</i>	199
J.P. Rosen <i>ASIS vs. LibAdalang: A Comparative Assesment</i>	203

Produced by Ada-Europe

---

## Editor in Chief

**António Casimiro**

University of Lisbon, Portugal  
*AUJ\_Editor@Ada-Europe.org*

---

## Ada User Journal Editorial Board

**Luís Miguel Pinho**  
*Associate Editor*

Polytechnic Institute of Porto, Portugal  
*lmp@isep.ipp.pt*

**Jorge Real**  
*Deputy Editor*

Universitat Politècnica de València, Spain  
*jorge@disca.upv.es*

**Patricia López Martínez**  
*Assistant Editor*

Universidad de Cantabria, Spain  
*lopezpa@unican.es*

**Kristoffer N. Gregertsen**  
*Assistant Editor*

SINTEF, Norway  
*kristoffer.gregertsen@sintef.no*

**Dirk Craeynest**  
*Events Editor*

KU Leuven, Belgium  
*Dirk.Craeynest@cs.kuleuven.be*

**Alejandro R. Mosteo**  
*News Editor*

Centro Universitario de la Defensa, Zaragoza, Spain  
*amosteo@unizar.es*

---

## Ada-Europe Board

**Tullio Vardanega** (President)  
University of Padua

Italy

**Dirk Craeynest** (Vice-President)  
Ada-Belgium & KU Leuven

Belgium

**Dene Brown** (General Secretary)  
SysAda Limited

United Kingdom

**Ahlan Marriott** (Treasurer)  
White Elephant GmbH

Switzerland

**Luís Miguel Pinho** (Ada User Journal)  
Polytechnic Institute of Porto

Portugal

**António Casimiro** (Ada User Journal)  
University of Lisbon

Portugal



---

## Ada-Europe General Secretary

Dene Brown  
SysAda Limited  
Signal Business Center  
2 Innotec Drive  
BT19 7PD Bangor  
Northern Ireland, UK

Tel: +44 2891 520 560  
Email: [Secretary@Ada-Europe.org](mailto:Secretary@Ada-Europe.org)  
URL: [www.ada-europe.org](http://www.ada-europe.org)

---

## Information on Subscriptions and Advertisements

Ada User Journal (ISSN 1381-6551) is published in one volume of four issues. The Journal is provided free of charge to members of Ada-Europe. Library subscription details can be obtained direct from the Ada-Europe General Secretary (contact details above). Claims for missing issues will be honoured free of charge, if made within three months of the publication date for the issues. Mail order, subscription information and enquiries to the Ada-Europe General Secretary.

For details of advertisement rates please contact the Ada-Europe General Secretary (contact details above).

---

# ADA USER JOURNAL

Volume 42  
Numbers 3-4  
September –  
December 2021

---

## Contents

	<i>Page</i>
Editorial Policy for <i>Ada User Journal</i>	128
Editorial	129
Quarterly News Digest	130
Conference Calendar	170
Forthcoming Events	179
Articles from the AEiC 2021 Work-In-Progress Session	
K. Nyborg Gregertsen <i>“Ember: An Embedded Robotics Library in SPARK”</i>	185
D. García Villaescusa, M. Aldea Rivas, M. González Harbour <i>“Queuing Ports for Mesh Based Many-Core Processors”</i>	189
AEiC 2021 Industrial Presentations	
A. Marriot, U. Maurer <i>“More Ada in Non-Ada Systems”</i>	193
M. Martignano <i>“Static Analysis for Ada, C/C++ and Python: Different Languages, Different Needs”</i>	199
J-P. Rosen <i>“ASIS vs. LibAdalang: A Comparative Assesment”</i>	203
Ada-Europe Associate Members (National Ada Organizations)	208
Ada-Europe Sponsors	Inside Back Cover



# Editorial Policy for Ada User Journal

## Publication

*Ada User Journal* — The Journal for the international Ada Community — is published by Ada-Europe. It appears four times a year, on the last days of March, June, September and December. Copy date is the last day of the month of publication.

## Aims

*Ada User Journal* aims to inform readers of developments in the Ada programming language and its use, general Ada-related software engineering issues and Ada-related activities. The language of the journal is English.

Although the title of the Journal refers to the Ada language, related topics, such as reliable software technologies, are welcome. More information on the scope of the Journal is available on its website at [www.ada-europe.org/auj](http://www.ada-europe.org/auj).

The Journal publishes the following types of material:

- Refereed original articles on technical matters concerning Ada and related topics.
- Invited papers on Ada and the Ada standardization process.
- Proceedings of workshops and panels on topics relevant to the Journal.
- Reprints of articles published elsewhere that deserve a wider audience.
- News and miscellany of interest to the Ada community.
- Commentaries on matters relating to Ada and software engineering.
- Announcements and reports of conferences and workshops.
- Announcements regarding standards concerning Ada.
- Reviews of publications in the field of software engineering.

Further details on our approach to these are given below. More complete information is available in the website at [www.ada-europe.org/auj](http://www.ada-europe.org/auj).

## Original Papers

Manuscripts should be submitted in accordance with the submission guidelines (below).

All original technical contributions are submitted to refereeing by at least two people. Names of referees will be kept confidential, but their comments will be relayed to the authors at the discretion of the Editor.

The first named author will receive a complimentary copy of the issue of the Journal in which their paper appears.

By submitting a manuscript, authors grant Ada-Europe an unlimited license to publish (and, if appropriate, republish) it, if and when the article is accepted for publication. We do not require that authors assign copyright to the Journal.

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

## Proceedings and Special Issues

The *Ada User Journal* is open to consider the publication of proceedings of workshops or panels related to the Journal's aims and scope, as well as Special Issues on relevant topics.

Interested proponents are invited to contact the Editor-in-Chief.

## News and Product Announcements

Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will reprint or report on items that may be of interest to them.

## Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a

wider audience. This includes papers published in North America that are not easily available in Europe.

We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal*.

## Commentaries

We publish commentaries on Ada and software engineering topics. These may represent the views either of individuals or of organisations. Such articles can be of any length – inclusion is at the discretion of the Editor.

Opinions expressed within the *Ada User Journal* do not necessarily represent the views of the Editor, Ada-Europe or its directors.

## Announcements and Reports

We are happy to publicise and report on events that may be of interest to our readers.

## Reviews

Inclusion of any review in the Journal is at the discretion of the Editor. A reviewer will be selected by the Editor to review any book or other publication sent to us. We are also prepared to print reviews submitted from elsewhere at the discretion of the Editor.

## Submission Guidelines

All material for publication should be sent electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.



# Editorial

I would like to start this editorial with a note about this issue being a double issue. The decision of merging the September and December issues of the AUJ into a single issue (AUJ 42-3-4) was mainly motivated by the fact that we wanted to get back on track concerning the timeliness of the production and delivery of the journal. Our readers certainly understand that the COVID-19 pandemic, which continued throughout 2021, had a negative impact on the number of events taking place and hence on the amount of publishable material. Therefore, while we were able to solve the problems faced in 2020 concerning printing the journal, we struggled to collect good articles to include in the journal.

I would also like to report that we will continue sending the AUJ copies directly from Portugal to all subscribers, leveraging the experience with the last issue, which we believe was very positive.

In this issue we conclude the publication of the proceedings of the AEiC 2021 Work-In-Progress (WiP) Session and include three papers that are extended versions of Industrial Presentations given also at AEiC 2021. In concrete, the first paper is entitled “Ember: An Embedded Robotics Library in SPARK” and is authored by K. Gregertsen, from SINTEF Digital, in Norway. The paper presents a library named Ember, which is intended for high-integrity embedded robotics and GNC applications developed in SPARK 2014 with formal verification. Then, the WiP Session proceedings are closed with a contribution from the University of Cantabria, an article entitled “Queuing Ports for Mesh Based Many-Core Processors”, authored by D. G. Villaescusa, M. A. Rivas and M. G. Harbour. The article describes the implementation of Queuing Ports, which are a communication mechanism for many-core architectures to allow tasks running in different cores to communicate in a synchronized fashion. The three papers derived from Industrial Presentations follow. Firstly, a paper entitled “More Ada in Non-Ada Systems”, by A. Marriot and U. Maurer, from White Elephant GmbH in Switzerland, which advocates the relevance of using code written in Ada to supplement existing code written in other languages. Then, M. Martignano, from Spazio IT in Italy, writes about “Static analysis for Ada, C/C++ and Python: Different Languages, Different Needs”. Finally, J-P. Rosen, from Adalog in France, contributes with a paper entitled “ASIS vs. LibAdalang: A Comparative Assesment”, which “compares the origins, features, and status of two different tools intended to facilitate static analysis of Ada programs: ASIS and LibAdalang”, as expressed by the author.

The News Digest and the Calendar and Events sections are included as usual, but in this issue they cover a longer period, corresponding to the second half of 2021. As usual, these sections are prepared, respectively, by Alejandro R. Mosteo and by Dirk Craeynest. Last but not the least, I would like to call your particular attention to an article that we include as Forthcoming Event announcement, which calls the Ada community to participate in the Google Summer of Code (GSoC) programme. This article was prepared by Fernando Oleo Blanco, an open source and Ada enthusiast, and provides information on how to participate.

*Antonio Casimiro  
Lisboa  
December 2021  
Email: AUJ\_Editor@Ada-Europe.org*

# 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

Preface by the News Editor	130
Ada-related Events	130
Ada and Education	131
Ada-related Resources	133
Ada-related Tools	141
Ada-related Products	146
Ada and Operating Systems	147
Ada and Other Languages	150
Ada Practice	153

[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 an 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](http://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](http://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<sup>1</sup>) members [2]
- Stack Overflow: 2\_125 (+38) questions [3]
- Libera.Chat<sup>2</sup>: 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]

<sup>1</sup> Probably caused in part by confusion with the ADA cryptocurrency.

<sup>2</sup> 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](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<sup>1</sup> (=) 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]

<sup>1</sup> This number is unreliable due to GitHub search limitations.

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

[2] [http://rosettacode.org/wiki/Category:Ada\\_User](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](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

Alslys. 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 libadaling, if there is a need for that.

I also suggested that libadaling 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 libadaling, 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 LibAdaling 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 LibAdaling.

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 libadaling.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 LibAdaling 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 LibAdaling 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 libadaling, 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 items 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 Brukar dt  
<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 Brukar dt  
<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](https://www.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 [-gnatvw]
wisitoken-syntax_trees.ads:620:04:
warning: aggregate not fully initialized
[-gnatvw]
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/c75234037fb45
68739435fad204f206afe609a77
```

*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" wis3.1.3,

Sigh. to be fixed in 7.1.8.

I could argue that wis3.1.3 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.aioc.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_ListClass;
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/](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 Xnadolib) 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](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 11.5.2.

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 11.5.2.

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](mailto: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](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](mailto: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](https://gitlab.com/daresoft/network/adare_net)

*From: Drpi* <[314@drpi.fr](mailto: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*  
<[joakims@kth.se](mailto:joakims@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](mailto: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*  
<[joakims@kth.se](mailto:joakims@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](mailto: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](mailto: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*

*Newsroups: 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@nosspam.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://vmsoftware.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://vmsoftware.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](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](http://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](mailto: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](mailto: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](mailto: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/c550eafe889691af212379590974944e1359e180/gcc10\\_aux](https://github.com/NetBSD/pkgsrc-wip/tree/c550eafe889691af212379590974944e1359e180/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-topmo.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? EINTR? 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 |
  EINTR);
```

EINTR 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*

```
> EINTR 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 Eratosthenes searching for primes under 1 million.

Odd number search only, can be multithreaded and skip ahead. See the rules at: <https://github.com/plummersoftwarellc/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](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](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  
<m8illists@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/](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  
<joakims@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  
<joakims@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@tdorum.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](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](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](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 an 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@nosпам.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](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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@nosspam.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@nosspam.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-afplusplus-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](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/vzajq2ym10w/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/55eeeb08aedecb68fd812f/RAPID-A-Free-Portable-GUI-Design-Tool](https://www.researchgate.net/profile/Martin-Carlisle/publication/221444571_RAPID_A_Free_Portable_GUI_Design_Tool/links/55eeeb08aedecb68fd812f/RAPID-A-Free-Portable-GUI-Design-Tool)

CLAW:

<http://www.rsoftware.com/html/prodinf/claw/claw.htm>

Paper:

<http://www.rsoftware.com/html/prodinf/triadapaper/triada.html>

JEWEL:

<http://archive.adaic.com/tools/bindings/JEWEL/jewel-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](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](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/](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_Test :=
   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@rsoftware.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 : 9.99900000000000000000E+01
Image_Odd_2 : 9.99900000000000000000E+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.fi>  
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](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/](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 automatically 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 reimplemention 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](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:

```
CharacterVal (16#E2#) &
CharacterVal (16#82#) &
CharacterVal (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 cyrillic 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",
  Blurp => "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 for 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 :(



ptc® apexada | ptc® objectada®

# Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: [ptc.com/developer-tools](http://ptc.com/developer-tools)



# Conference Calendar

**Dirk Craeynest**

*KU Leuven, Belgium. Email: Dirk.Craeynest@cs.kuleuven.be*

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☺ denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at <http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html> on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. Where available, the status of events is indicated with the following markers: "(v)" = event is held online, "(h)"= event is held in a hybrid form (i.e. partially online).

---

## 2021

- October 3-7  
(h) **16th International Conference on Software Engineering Advances (ICSEA'2021)**. Barcelona, Spain. Topics include: trends and achievements; advances in fundamentals for software development (software analysis and model checking, software design, design by contract, software modeling, software validation and verification, software testing and testing tools, software implementation, component-based software development, software security-based development, ...); advanced mechanisms for software development (software composition, refactoring, software dependencies, software rejuvenation, embedded software, parallel and distributed software, ...); advanced design tools for developing software (formal specifications in software, programming mechanisms such as real-time, multi-threads, etc., programming techniques, programming languages, ...); software security, privacy, safeness (software safety and security, software vulnerabilities, assessing risks in software, software for online banking and transactions, high confidence software, ...); advances in software testing; specialized software advanced applications (software for mobile vehicles, biomedical-related software, mission critical software, real-time software, e-health related software, ...); open source software; agile and lean approaches in software engineering; software deployment and maintenance; software engineering techniques, metrics, and formalisms (software reuse, software re-engineering, software composition, software integration, ...); software economics, adoption, and education; etc.
- October 8-15  
(v) **Embedded Systems Week 2021 (ESWEEK'2021)**. Shanghai, China. The venues for ESWEEK 2020 and 2021 were swapped. ESWEEK 2020 was to be held in Hamburg, Germany from September 20-25, 2020. ESWEEK 2021 would be held in Shanghai, China from October 10-15, 2021, but then moved to a virtual event format. Includes CASES'2021 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2021 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2021 (International Conference on Embedded Software).
- October 10-15  
(v) **ACM SIGBED International Conference on Embedded Software (EMSOFT'2021)**. Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which integrate computation, networking, and physical dynamics.
- October 10-15  
(v) **International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'2021)**. Topics include: system-level design, hardware/software co-design, modeling, analysis, and implementation of modern Embedded Systems, Cyber-Physical Systems, and Internet-of-Things, from system-level specification and optimization to system synthesis of multi-processor hardware/software implementations.
- October 10-15  
(v) **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'2021)**. Topics include: latest advances in compilers and architectures for high-performance, low-power, and domain-specific embedded systems; compilers for embedded systems: multi- and many-core processors, GPU architectures,

reconfigurable computing including FPGAs and CGRAs, security, reliability, and predictability (secure architectures, hardware security, and compilation for software security; architecture and compiler techniques for reliability and aging; modeling, design, analysis, and optimization for timing and predictability; validation, verification, testing & debugging of embedded software); etc.

- October 11-14  
(h) 21st **International Conference on Runtime Verification (RV'2021)**. Los Angeles, California, USA. Topics include: monitoring and analysis of runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission critical systems, enterprise and systems software, cloud systems, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy, among others.
- October 11-15  
(h) 15th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement (ESEM'2021)**. Bari, Italy. ESEM'2020 was postponed from 8-9 October 2020 to 2021.
- ☺ October 17-22  
(h) **ACM Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH'2021)**. Chicago, Illinois, USA. Topics include: all aspects of software construction and delivery, at the intersection of programming, languages, and software engineering.
- October 17-19 14th ACM SIGPLAN **International Conference on Software Language Engineering (SLE'2021)**. Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; software language engineering rather than engineering a specific software language; software language design and implementation; software language validation; software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.
- October 17-22 28th **Static Analysis Symposium (SAS'2021)**. Chicago, Illinois, USA. In conjunction with SPLASH'2021. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.
- October 18-22  
(v) 19th **International Symposium on Automated Technology for Verification and Analysis (ATVA'2021)**. Gold Coast, Australia. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware, software, and machine learning (ML) systems; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; verification in industrial practice; applications and case studies; automated tool support; etc.
- October 25  
(v) 11th **Workshop on Programming Languages and Operating Systems (PLOS'2021)**. Internet. Topics include: critical evaluations of new programming language ideas in support of OS construction; type-safe languages for operating systems; language-based approaches to crosscutting system concerns, such as security and run-time performance; language support for system verification, testing, and debugging; the use of OS abstractions and techniques in language runtimes; experience reports on applying new language techniques in commercial OS settings; etc.
- October 28-29  
(v) 17th **International Conference on Formal Aspects of Component Software (FACS'2021)**. Grenoble, France. Topics include: application of formal methods in all aspects of software components and services, such as formal models for software components and their interaction; design and verification methods for software components and services; formal methods and modeling languages for components and services; components for real-time, safety-critical, secure, and/or embedded systems; components for the Internet of things and cyber-physical systems; model-based testing of components and services; case studies and experience reports; tools supporting formal methods for components and services; etc.
- November 01-02 24th **International Workshop on Software and Compilers for Embedded Systems (SCOPES'2021)**. Eindhoven, the Netherlands. Topics include: all aspects of the compilation and mapping process of embedded systems, such as models of computation and programming languages, automatic code parallelization techniques, mapping and scheduling techniques for embedded multi-processor systems, code generation techniques for embedded single- and multi-processor architectures, design of real-time



systems, techniques for compiler aided profiling, measurement, debugging and validation of embedded software, etc.

- November 15-19  
(v) 36th **IEEE/ACM International Conference on Automated Software Engineering (ASE'2021)**. Melbourne, Australia. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems; testing, verification, and validation; software analysis; empirical software engineering; maintenance and evolution; software security and trust; program comprehension; software architecture and design; reverse engineering and re-engineering; model-driven development; specification languages; software product line engineering; etc.
- November 17-20  
(v) 23rd **International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'2021)**. Internet. Topics include: foundations of concurrent and distributed computing; distributed and concurrent algorithms and data structures; synchronization protocols; formal methods, validation, verification, and synthesis; fault tolerance, security, and privacy; verifiable/fault-tolerant computing; etc.
- November 20-26  
(v) 24th **International Symposium on Formal Methods (FM'2021)**. Beijing, China. Topics include: formal methods in a wide range of domains including software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, and healthcare; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, experiments with challenge problems); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, and experimental validation of tools); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, and method integration); etc. Deadline for submissions: October 4, 2021 (Doctoral Symposium research abstracts).
- November 22-23  
(v) 15th **International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS'2021)**. Beijing, China. Topics include: formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems; abstraction techniques; compositional verification; correct-by-construction design; rigorous system design; model-checking; performance and robustness evaluation; QoS evaluation, planning and deployment; dependability assessment techniques; RAMS (Reliability-Availability-Maintainability-Safety) assessment; model-based security assessment; verification & validation of IoT and of safety-critical systems; assessment for real-time systems; worst-case execution time analysis; etc. Application areas include: communication protocols, cyber-physical systems, high-performance computing, internet of things, logistics systems, mixed criticality systems, programming languages, real-time and embedded operating systems, telecommunication systems, etc.
- November 25-26 22nd **International Conference on Product-Focused Software Process Improvement (PROFES'2021)**. Turin, Italy. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs.
- December 01-04  
(v) 25th **Pacific Rim International Symposium on Dependable Computing (PRDC'2021)**. Perth, Australia. Topics include: software and hardware reliability, testing, verification, and validation; dependability measurement, modeling, evaluation, and tools; software aging and rejuvenation; safety-critical and mixed-criticality systems and software; architecture and system design for dependability; (industrial) Internet of Things dependability, security and privacy; dependability issues in high performance computing, in real-time systems; in cyber-physical systems; dependability and security in AI and machine learning systems; etc.
- December 06-08  
(v) 20th **Belgium-Netherlands Software Evolution Workshop (BENEVOL'2021)**. 's-Hertogenbosch, the Netherlands. Topics include: software evolution and maintenance. Deadline for submissions: November 5, 2021 (technical papers), November 12, 2021 (presentation abstracts). Deadline for registration: November 22, 2021.
- December 06-09  
(v) 28th **Asia-Pacific Software Engineering Conference (APSEC'2021)**. Taiwan. Topics include: agile methodologies; component-based software engineering; cyber-physical systems and Internet of Things; debugging and fault localization; embedded real-time systems; formal methods; middleware, frameworks, and APIs; model-driven and domain-specific engineering; open source development; parallel, distributed, and concurrent systems; programming languages and systems; refactoring; reverse

engineering; security, reliability, and privacy; software architecture, modelling and design; software comprehension and traceability; software engineering education; software engineering tools and environments; software maintenance and evolution; software product-line engineering; software reuse; software repository mining; testing, verification, and validation; etc. Deadline for submissions: October 7, 2021 (poster papers).

- December 06-10  
(v) 19th **International Conference on Software Engineering and Formal Methods (SEFM'2021)**. York, UK. Topics include: software development methods (formal modeling, specification, and design; software evolution, maintenance, re-engineering, and reuse); design principles (programming languages, domain-specific languages, abstraction and refinement, ...); software testing, validation, and verification (model checking, theorem proving, and decision procedures; testing and runtime verification; lightweight and scalable formal methods; resilience, security, privacy, and trust; safety-critical, fault-tolerant, and secure systems; software assurance and certification; ...); applications and technology transfer (component-based and multi-agent systems; real-time, hybrid, and cyber-physical systems; education; ...); special topic "Software Engineering and Formal Methods for Resilient and Trustworthy Autonomous Systems"; case studies, best practices, and experience reports.
- December 06-14  
(h) 21st IEEE **International Conference on Software Quality, Reliability and Security (QRS'2021)**. Hainan Island, China. Topics include: reliability, security, availability, and safety of software systems; software testing, verification, and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; operating system security and reliability; benchmark, tools, industrial applications, and empirical studies; etc. Deadline for submissions: October 01, 2021 (fast abstracts, industry track, posters).
- ☺ Dec 07-10  
(v) 42nd IEEE **Real-Time Systems Symposium (RTSS'2021)**. Dortmund, Germany. RTSS'2021 was moved from Taipei, Taiwan, to Dortmund, Germany. Topics include: addressing some form of real-time requirements such as deadlines, response times or delays/latency; real-time system track (middleware, compilers, tools, scheduling, QoS support, testing and debugging, design and verification, modeling, WCET analysis, performance analysis, fault tolerance, security, system experimentation and deployment experiences, ...); design and application track (cyber-physical systems design methods, tools chains, security and privacy, performance analysis, robustness and safety, analysis techniques and tools, ...; architecture description languages and tools; Internet of Things (IoT) aspects of scalability, interoperability, reliability, security, middleware and programming abstractions, protocols, modelling, analysis and performance evaluation, ...); etc. Deadline for submissions: October 1, 2021 (TCRTS Test of Time Award nominations), October 29, 2021 (student travel grant applications). Deadline for early registration: October 27, 2021 (physical participation), December 10, 2021 (virtual participation).
- December 07-10  
(v) 24th **Brazilian Symposium on Formal Methods (SBMF'2021)**. Campina Grande, PB, Brazil. Topics include: development, dissemination, and use of formal methods for the construction of high-quality computational systems; applications of formal methods to software design, development, code generation, testing, maintenance, evolution, reuse, ...; specification and modelling languages (logic and semantics for specification or/and programming languages; formal methods for timed, real-time, hybrid, or/and safety-critical systems; formal methods for cyber-physical systems; ...); theoretical foundations (type systems models of concurrency, security, ...); verification and validation (abstraction, modularization or/and refinement techniques, static analysis, model checking, theorem proving, software certification, correctness by construction); experience reports on teaching formal methods, on industrial application of formal methods.
- December 10  
Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!
- December 13-14  
(v) 17th **International Conference on Embedded Software and Systems (ICCESS'2021)**. Shanghai, P. R. China. Topics include: systems, models & algorithms track (real-time embedded systems, fault tolerant and trusted embedded systems, mixed-criticality embedded systems, multicore embedded systems, ...), design methodology & tools track (formal methods for embedded systems, middleware for embedded systems, IDE and software tools, verification and validation for embedded systems, compilation and debug techniques and tools, safety of machine learning for embedded systems, ...), emerging embedded applications and interdisciplinary topics track (machine learning for embedded applications, Internet-of-Things (IoT), robotics and control systems, Cyber-Physical Systems (CPS), automotive and avionics systems, medical systems, industrial practices and case studies, ...), etc.

---

**2022**

- January 17-19      **17th International Conference on High Performance and Embedded Architecture and Compilation (HiPEAC'2022)**. Budapest, Hungary. Topics include: computer architecture, programming models, compilers and operating systems for embedded and general-purpose systems.
- January 18-20      **14th Software Quality Days Conference and Tools Fair (SWQD'2022)**, Vienna, Austria. Topics include: all topics about software and systems quality, such as improvement of software development methods and processes, testing and quality assurance of software and software-intensive systems, project and risk management, domain specific quality issues (among others embedded, medical, automotive systems), novel trends in software quality, etc.
- ♦ February 6  
(v)      **11th Ada Developer Room at FOSDEM 2022**, Brussels, Belgium. FOSDEM 2022 is a two-day event (Sat-Sun 5-6 Feb), exceptionally held fully online. This years' edition includes once more a full-day Ada Developer Room, held on Sunday 6 February, and organized in cooperation with Ada-Belgium and Ada-Europe. Deadline for submissions: December 26, 2021 (initial presentation proposals).
- © February 12-16  
(h)      **26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'2022)**. Seoul, South Korea. Deadline for submissions: November 5, 2021 (workshops, tutorials).
- March 09-11  
(h)      **30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'2022)**. Valladolid, Spain.
- March 12-15  
(h)      **18th IEEE International Conference on Software Architecture (ICSA'2022)**. Honolulu, Hawaii, USA. Topics include: architecture evaluation and quality aspects of software architectures; model-driven engineering and component-based software engineering; automatic extraction and generation of software architecture descriptions; refactoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; linking architecture to requirements and/or implementation; architecture conformance; reusable architectural solutions; software architecture for legacy systems and systems integration; architecting families of products; roles and responsibilities for software architects; training, soft skills, coaching, mentoring, education, and certification of software architects; resilient and dependable software architectures; etc. Deadline for submissions: October 31, 2021 (student volunteers), November 1, 2021 (technical track abstracts), November 8, 2021 (technical track full papers), December 1, 2021 (Software Architecture in Practice track, New and Emerging Ideas track abstracts, Early Career Researchers Forum abstracts, poster track abstracts), December 8, 2021 (journal-first track, New and Emerging Ideas track full papers, Early Career Researchers Forum full papers, poster track full papers, tutorials), January 11, 2022 (Artifact Evaluation track mandatory registrations), January 14, 2022 (Artifact Evaluation track submissions).
- March 15-18      **29th IEEE Conference on Software Analysis, Evolution, and Reengineering (SANER'2022)**. Honolulu, Hawaii, USA. Topics include: theory and practice of recovering information from existing software and systems; software analysis, parsing, and fact extraction; software reverse engineering and reengineering; program comprehension; software evolution analysis; software architecture recovery and reverse architecting; program transformation and refactoring; mining software repositories and software analytics; software maintenance and evolution; software release engineering, continuous integration and delivery; education related to all of the above topics; etc. Deadline for submissions: October 10, 2021 (workshops), October 14, 2021 (main research track abstracts), October 21, 2021 (main research track papers), November 11, 2021 (ERA track abstracts, tool track and industry track abstracts, RENE track abstracts), November 18, 2021 (ERA track papers, tool track and industry track papers, RENE track papers), November 19, 2021 (journal first track), December 17, 2021 (workshop papers).
- March 21-24      **28th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'2022)**. Aston, Birmingham, UK. Theme: "Explainability in Requirements Engineering". Deadline for submissions: October 11, 2021 (workshops), October 18, 2021 (research paper abstracts), October 25, 2021 (full research papers), January 10, 2022 (poster and tool abstracts), January 17, 2022 (workshop papers, posters, tools), January 28, 2022 (doctoral symposium).

- ☺ March 21-25 (h) **International Conference on the Art, Science, and Engineering of Programming (Programming'2022)**. Porto, Portugal. Deadline for submissions: December 5, 2021 (workshops).
- ☺ March 30-31 11th **European Congress on Embedded Real Time Systems (ERTS'2022)**. Toulouse, France. Topics include: all aspects of critical embedded real-time systems, such as model-based system engineering, formal methods, product line engineering, new programming and verification languages, dependability, safety, cyber security, quality of service, fault tolerance, maintainability, certification, robotics, etc. Deadline for submissions: October 3, 2021 (regular abstracts, short papers), January 9, 2022 (regular papers), January 30, 2022 (final short and regular papers).
- April 02-03 31st ACM SIGPLAN **International Conference on Compiler Construction (CC'2022)**. Seoul, South Korea. Co-located with CGO, HPCA, and PPOPP. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (including program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (including memory management, virtual machines, and dynamic and just-in-time compilation); programming tools (including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques, ranging from programming languages to micro-architectural support, for specific domains such as secure, parallel, distributed, embedded or mobile environments; design and implementation of novel language constructs, programming models, and domain-specific languages. Deadline for submissions: November 8, 2021 (full papers), February 7, 2022 (artifacts).
- April 02-07 25th **European Joint Conferences on Theory and Practice of Software (ETAPS'2022)**. Munich, Germany. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems). Deadline for submissions: October 14, 2021 (papers).
- April 04-13 (v) 15th IEEE **International Conference on Software Testing, Verification and Validation (ICST'2022)**. Valencia, Spain. Deadline for submissions: December 17, 2021 (tool demos), January 8, 2022 (doctoral symposium), February 4, 2022 (posters).
- April 09-13 13th ACM/SPEC **International Conference on Performance Engineering (ICPE'2022)**. Beijing, China.
- April 23 (h) 2nd **International Conference on Code Quality (ICCQ'2022)**. Innopolis, Kazan, Russia. Topics include: static analysis, program verification, bug detection, and software maintenance. Deadline for submissions: December 18, 2021 (abstracts, papers).
- April 25-29 (h) 37th ACM **Symposium on Applied Computing (SAC'2022)**. Brno, Czech Republic.
- April 25-29 17th **Track on Dependable, Adaptive, and Secure Distributed Systems (DADS'2022)**. Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc. Deadline for paper submissions: October 15, 2021.
- May 03-06 15th **Cyber-Physical Systems and Internet of Things Week (CPS Week'2022)**. Milan, Italy. Event includes: 5 top conferences, HSCC, ICCPS, IPSN, RTAS, and IoTDI, multiple workshops, tutorials, competitions and various exhibitions from both industry and academia.
- ☺ May 04-06 CPSWeek2022 - 28th IEEE **Real-Time and Embedded Technology and Applications Symposium (RTAS'2022)**- Topics include: systems research related to embedded systems and time-sensitive systems; original systems, applications, case studies, methodologies, and algorithms that contribute to the state of practice in design, implementation, verification, and validation of embedded systems or time-sensitive systems. Deadline for submissions: October 29, 2021 (papers).
- ☺ May 17-19 25th IEEE **International Symposium On Real-Time Distributed Computing (ISORC'2022)**. Västerås, Sweden. Topics include: all aspects of object/component/service-oriented real-time distributed computing (ORC) technology, such as distributed computing, internet of things (IoT), real-time scheduling theory, resilient cyber-physical systems, autonomous systems (e.g., autonomous driving), optimization of time-sensitive applications, applications based on ORC technology (e.g., medical



devices, intelligent transportation systems, industrial automation systems and industry 4.0, smart grids, ...), etc. Deadline for submissions: January 23, 2022 (regular papers).

- May 21-29      **44th International Conference on Software Engineering (ICSE'2022)**. Pittsburgh, Pennsylvania, USA. Topics include: the full spectrum of Software Engineering. Deadline for submissions: October 15, 2021 (NIER - New Ideas and Emerging Results, SEIP - Software Engineering in Practice), October 22, 2021 (SEET - Software Engineering Education and Training, SEIS - Software Engineering in Society), November 24, 2021 (demos), January 12, 2022 (Journal-First papers), January 14, 2022 (workshop papers).
- May 22-23      **5th International Conference on Technical Debt (TechDebt'2022)**. Deadline for submissions: January 13, 2022 (main research track abstracts), January 18, 2022 (main research track papers), January 31, 2022 (tool presentation papers), March 30, 2022 (tool track extended abstracts).
- May 24-27  
(h)      **14th NASA Formal Methods Symposium (NFM'2022)**. Pasadena, California, USA. Topics include: challenges and solutions for achieving assurance for critical systems, such as advances in formal methods (interactive and automated theorem proving, model checking, static analysis, runtime verification, automated testing, design for verification and correct-by-design techniques, ...), integration of formal methods techniques, formal methods in practice (experience reports of application of formal methods on real systems, such as autonomous systems, safety-critical systems, concurrent and distributed systems, cyber-physical, embedded, and hybrid systems, ...; use of formal methods in education; reports on negative results in the development and the application for formal methods in practice; usability of formal method tools, and their infusion into industrial contexts; ...).
- ☺ June 01-02      **International Conference on Reliability, Safety and Security of Railway Systems (RSSRail'2022)**. Paris, France. Topics include: building critical railway applications and systems; safety in development processes and safety management; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, toolchains; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc. Deadline for submissions: November 25, 2021 (abstracts), December 2, 2021 (full papers).
- ☺ June 06-10      **36th European Conference on Object-Oriented Programming (ECOOP'2022)**. Berlin, Germany. Topics include: all practical and theoretical investigations of programming languages, systems and environments; innovative solutions to real problems as well as evaluations of existing solutions. Deadline for submissions: December 1, 2021 (round 1 submissions), December 10, 2021 (round 1 artifacts), December 15, 2021 (nominations for Dahl-Nygaard prizes), March 1, 2022 (round 2 submissions), March 10, 2022 (round 2 artifacts).
- June 07-10      **17th International Conference on integrated Formal Methods (iFM'2022)**. Lugano, Switzerland. Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support and the use of such techniques in software engineering practice. Deadline for submissions: January 14, 2022 (abstracts), January 21, 2022 (papers), March 28, 2022 (artefacts).
- ☺ June 08-09  
(h)      **30th International Conference on Real-Time Networks and Systems (RTNS'2022)**. Paris, France. Topics include: real-time application design and evaluation (automotive, avionics, space, railways, telecommunications, process control, ...), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, security, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), ...), formal specification and verification, real-time distributed systems, etc. Deadline for submissions: February 24, 2022.
- June 13-15  
(h)      **26th International Conference on Empirical Assessment and Evaluation in Software Engineering (EASE'2022)**. Gothenburg, Sweden. Topics include: assessing the benefits/costs associated with using chosen development technologies; empirical studies using qualitative, quantitative, and mixed methods; evaluation and comparison of techniques and models; modeling, measuring, and assessing product and/or process quality; replication of empirical studies and families of studies; software technology

transfer to industry; etc. Deadline for submissions: November 8, 2021 (workshops, tutorials), December 6, 2021 (paper abstracts), December 12, 2021 (full papers), January 5, 2022 (short papers and artifacts), January 10, 2022 (workshop and tutorial papers), January 24, 2022 (doctoral symposium papers), February 12, 2022 (industry experience reports), February 17, 2022 (vision papers and emerging results).

- June 13-17      25th **Ibero-American Conference on Software Engineering (CIbSE'2022)**. Cordoba, Argentina. Topics include: formal methods applied to software engineering (SE), mining software repositories and software analytics, model-driven SE, software architecture, software dependability, SE education and training, SE for emerging application domains (e.g., cyber-physical systems, IoT, ...), SE in industry, software maintenance and evolution, software process, software product lines, software quality and quality models, software reuse, software testing, technical debt management, etc. Deadline for submissions: February 7, 2022 (abstracts), February 14, 2022 (papers), March 8, 2022 (doctoral symposium, journal first).
- ♦ June 14-17  
(h)      26th **Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022 aka Ada-Europe 2022)**, Ghent, Belgium. Sponsored by Ada-Europe. In cooperation with ACM SIGAda (pending), and the Ada Resource Association (ARA). Deadline for submissions: January 16, 2022 (journal-track papers, tutorials and workshop proposals), 27 February 2022 (industrial track and work-in-progress abstracts).
- June 15-17  
(h)      20th **International Conference on Software and Systems Reuse (ICSR'2022)**. Montpellier, France. Theme: "Reuse and Software Quality". Topics include: new and innovative research results and industrial experience reports dealing with all aspects of software reuse within the context of the modern software development landscape, such as technical aspects of reuse ( model-driven development, variability management and software product lines, domain-specific languages, new language abstractions for software reuse, software composition and modularization, ...), software reuse in industry (reuse success stories, reuse failures and lessons learned, reuse obstacles and success factors, return on Investment studies), etc. Deadline for submissions: December 1, 2021 (workshops), January 7, 2022 (paper abstracts), January 21, 2022 (full papers), January 28, 2022 (tutorials), March 10, 2022 (doctoral symposium), March 15, 2022 (tool demos).
- September 04-07      17th **Federated Conference on Computer Science and Information Systems (FedCSIS'2022)**. Sofia, Bulgaria. Deadline for submissions: November 21, 2022 (technical sessions).
- December 10      Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

# *Ada User Journal*

## Call for Contributions

Topics: **Ada, Programming Languages, Software Engineering Issues and Reliable Software Technologies** in general.

Contributions: **Refereed Original Articles, Invited Papers, Proceedings** of workshops and panels and **News and Information** on Ada and reliable software technologies.

More information available on the  
Journal web page at



<http://www.ada-europe.org/auj>

Online archive of past issues at <http://www.ada-europe.org/auj/archive/>



# FOSDEM'22

## Call for Participation 11<sup>th</sup> Ada Developer Room at FOSDEM 2022 Sunday 6 February 2022, online from Brussels, Belgium

*Organized in cooperation with  
Ada-Belgium<sup>1</sup> and Ada-Europe<sup>2</sup>*

FOSDEM<sup>3</sup>, the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. The 2022 edition takes place on Saturday 5 and Sunday 6 February. It is free to attend and no registration is necessary. This year, for obvious reasons, it has been turned into an online event, just like last year.

In this edition, the Ada FOSDEM community organizes once more 8 hours of presentations related to Ada and Free or Open Software in a s.c. Developer Room. The “Ada DevRoom” at FOSDEM 2022 is held on the 2<sup>nd</sup> day of the event, and offers introductory presentations on the Ada programming language, as well as more specialised presentations on focused topics, tools and projects: a total of 13 Ada-related presentations by 12 authors from 8 countries!

Program overview:

- *Introduction to the Ada DevRoom*, by Fernando Oleo Blanco, Germany
- *Introduction to Ada for Beginning and Experienced Programmers*, by Jean-Pierre Rosen, France
- *Ada Looks Good, Now Program a Game Without Knowing Anything*, by Stefan Hild, Germany
- *The Ada Numerics Model*, by Jean-Pierre Rosen, France
- *2022 Alire Update*, by Fabien Chouteau, France, Alejandro Mosteo, Spain
- *SweetAda: Lightweight Development Framework for Ada-based Software Systems*, by Gabriele Galeotti, Italy
- *Use (and Abuse?) of Ada 2022 Features to Design a JSON-like Data Structure*, by Alejandro Mosteo, Spain
- *Getting Started with AdaWebPack*, by Max Reznik, Ukraine
- *Overview of Ada GUI*, by Jeffrey Carter, Belgium
- *SPARKNaCl: a Verified, Fast Re-implementation of TweetNaCl*, by Roderick Chapman, UK
- *The Outsider's Guide to Ada: Lessons from Learning Ada in 2021*, by Paul Jarrett, USA
- *Proving the Correctness of the GNAT Light Runtime Library*, by Yannick Moy, France
- *Implementing a Build Manager in Ada*, by Stephane Carrez, France
- *Exporting Ada Software to Python and Julia*, by Jan Vershelde, USA
- *Closing of the Ada DevRoom*, by Dirk Craeynest, Belgium, Fernando Oleo Blanco, Germany

The Ada at FOSDEM 2022 web-page will have all details, such as the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info, including live video streaming and chat, plus recordings afterwards. For the latest information at any time, contact Fernando Oleo Blanco <irvise@irvise.xyz>, or see:

**<https://fosdem.org/2022/schedule/track/ada/>**  
**<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/22/220206-fosdem.html>**

<sup>1</sup><http://www.cs.kuleuven.be/~dirk/ada-belgium/>

<sup>2</sup><http://www.ada-europe.org/>

<sup>3</sup><https://fosdem.org/2022/>

# 26<sup>th</sup> Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022)

14-17 June 2022, Ghent, Belgium

## Conference Chair

*Tullio Vardanega*  
University of Padua, Italy  
tullio.vardanega@unipd.it

## Journal-track Chair

*Jérôme Hugues*  
Carnegie Mellon University, USA  
jjhugues@sei.cmu.edu

## Industrial-track Chair

*Alejandro R. Mosteo*  
Centro Universitario de la Defensa,  
Zaragoza, Spain  
amosteo@unizar.es

## Work-in-Progress-track Chair

*Frank Singhoff*  
University of Brest, France  
frank.singhoff@univ-brest.fr

## Tutorial and Workshop Chair

*Aurora Agar Armario*  
NATO, Netherlands  
aurora.agar@ncia.nato.int

## Exhibition & Sponsorship Chair

*Ahlan Marriott*  
White Elephant GmbH, Switzerland  
software@white-elephant.ch

## Publicity Chair

*Dirk Craeynest*  
Ada-Belgium & KU Leuven, Belgium  
dirk.craeynest@cs.kuleuven.be

## Local Chair

*Vicky Wandels*  
University of Ghent, Belgium  
Vicky.Wandels@UGent.be



In cooperation with  
ACM SIGAda (approval pending)  
and  
Ada Resource Association (ARA)

## General Information

The 26<sup>th</sup> [Ada-Europe International Conference on Reliable Software Technologies \(AEiC 2022\)](#) will take place in Ghent, Belgium in **dual mode**, with a solid core of in-presence activities accompanied by digital support for remote participation. The conference schedule comprises a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

## Schedule

- |                  |   |
|------------------|---|
| 16 January 2022  | Submission deadline for journal-track papers, tutorials and workshop proposals.   |
| 27 February 2022 | Submission deadline for industrial-track and work-in-progress-track abstracts.  |
| 14 March 2022    | Notification of invitations-to-present for journal-track papers.<br>Notification of acceptance for all other types of submission. |
| 3 April 2022     | Publication of advance program.   |

## Topics

The conference is an established international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of developing, running and maintaining challenging long-lived, high-quality software systems for a variety of application domains including manufacturing, robotics, avionics, space, health care, transportation, Cloud environments, smart energy, serious games. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software technologies.

The topics of interest for the conference include but are not limited to:

- [Real-Time and Safety-Critical Systems](#)
  - Design, implementation and verification challenges;
  - Novel approaches, e.g., Mixed-Criticality Systems, novel scheduling algorithms, novel design and analysis methods.
- [High-Integrity Systems and Reliability](#)
  - Theory and practice of High-Integrity Systems;
  - Languages vulnerabilities and countermeasures;
  - Architecture-centred development methods and tools
- [Reliability-oriented Programming Languages \(not limited to Ada\)](#)
  - Compilation and runtime challenges, language profiles;
  - Use cases and experience reports;
  - Language education and training initiatives.
- [Experience Reports](#)
  - Case studies, lessons learned, and comparative assessments;

Refer to the conference website for the full list of topics.



### Call for Journal-track Submissions

Following the [journal-first model](#) inaugurated in 2019, the conference includes a *journal-track* that seeks original and high-quality submissions that describe mature research work in the scope of the conference. Accepted papers for this track will be published in the "Reliable Software Technologies ([AEIC2022](#))" Special Issue of the *Journal of Systems Architecture* (JSA). General information for submitting to the JSA can be found at <https://www.journals.elsevier.com/journal-of-systems-architecture>. Submissions should be made online at <https://www.editorialmanager.com/jsa/> by selecting the "VSI:AEIC2022" option for the paper type.

In order to speed up publication, the JSA has adopted the Virtual Special Issue model, whereby acceptance decisions are made on a rolling basis. On that account, authors are encouraged to submit as early as they can, no later than 16 January 2022. *Authors who have successfully passed the first round of review will be invited to present their work at the conference.* Ada-Europe, the main conference sponsor, will cover the Open Access fees for the *first four papers* to gain final acceptance, which do not already enjoy OA from personalized bilateral agreements with the Publisher.

Prospective authors may direct all enquiries regarding this track to the corresponding chair, Jérôme Hugues, at the listed address.

### Call for Industrial-track Submissions

The conference seeks [industrial practitioner presentations](#) that deliver insight on the challenges of developing reliable software. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer review process. Interested authors shall submit a *short (one-to-two pages) abstract*, by 27 February 2022, via <https://easychair.org/conferences/?conf=aeic2022>, strictly in PDF, following the Ada User Journal style (cf. <http://www.ada-europe.org/auj/>).

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be invited to expand their contributions into full-fledged articles for publication in the *Ada User Journal*, which will form the proceedings of the Industrial track of the Conference.

Prospective authors may direct all enquiries regarding this track to the corresponding chair, Alejandro R. Mosteo, at the listed address.

### Call for Work-in-Progress-track Submissions

The Work-in-Progress track seeks two kinds of submissions: (a) [ongoing research](#), and (b) [early-stage ideas](#). *Ongoing research* submissions are 4-page papers that describe research results that are not mature enough to be submitted to the journal track as yet. *Early-stage ideas*, are 1-page papers that pitch new research directions that fall in the scope of the conference. Both kinds of submission must be original and shall undergo anonymous peer review. Submissions by recent MSc graduates and PhD students are especially sought. Authors shall submit their work by 27 February 2022, via <https://easychair.org/conferences/?conf=aeic2022>, strictly in PDF, following the Ada User Journal style (cf. <http://www.ada-europe.org/auj/>).

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be offered the opportunity to expand their contributions into 4-page articles for publication in the *Ada User Journal*, which will form the proceedings of the WIP track of the Conference.

Prospective authors may direct all enquiries regarding this track to the corresponding chair, Frank Singhoff, at the listed address.

### Academic Listing

The *Journal of Systems Architecture*, publication venue of the journal-track proceedings of the conference, was ranked Q1 (SJR) in the year 2020, also featuring 72<sup>th</sup> percentile in CiteScope (Scopus). The *Ada User Journal*, venue of all other technical proceedings of the conference, is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

### Awards

Ada-Europe will offer an honorary award for the best technical presentation, to be announced in the closing session of the conference.

### Call for Tutorials

The conference seeks [tutorials](#) in the form of educational seminars on themes falling within the conference scope, with an academic or practitioner slant, including hands-on or practical elements. Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), the intended level of the contents (introductory, intermediate, or advanced), and a statement motivating attendance. Tutorial proposals shall be submitted by e-mail to the Workshop and Tutorial Chair, Aurora Agar Armario, at the listed address, with subject line: "[AEIC 2022: tutorial proposal]".

The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The *Ada User Journal* will offer space for the publication of summaries of the accepted tutorials.

### Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference proper. Workshop proposals shall be submitted by e-mail to the Workshop and Tutorial Chair, Aurora Agar Armario, at the listed address, with subject line: "[AEIC 2022: workshop proposal]". Workshop organizers shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*.

### Call for Exhibitors

The conference will include a vendor and technology exhibition. Interested providers should direct inquiries to the Exhibition Chair.

### Venue

The conference will take place in the heart of the city of Ghent, Belgium, capital city of the East Flanders, north-west of Brussels, a half-hour train ride from it. Ghent is rich in history, culture and higher-education, with a top-100 university founded in 1817.

# Google Summer of Code (GSoC) and Ada Community Participation

*Fernando Oleo Blanco*

*Open Source & Ada aficionado; Tel: +34 689 44 27 45; email: irvise@irvise.xyz*

## Abstract

*Since 2005 the company known as Google (now part of the Alphabet conglomerate) has organized the “Google Summer of Code (GSoC)”, a program to fund work on Open Source projects. Since its beginnings, Google has offered the funding exclusively to students over the age of 18. However, in 2021 it was announced that the requirement to be a student was no longer in effect. This opens the door to a broader audience, which can benefit greatly the Ada community at large. This article will give some insights and information on how projects and organizations can participate on it.*

*Keywords: community, funding, Open Source, GSoC, Ada.*

## 1 GSoC Origins and Goals

Google, among other “big tech” companies, offers grants and funding to activities that it considers beneficial. Since 2005 one of its main projects has been the Google Summer of Code. It is a yearly event where organizations developing or involved in the development of Open Source Software, can propose a task for another person to do. Google, in turn, would select the projects that it deemed fit and would fund a volunteering student to work on such task.

In its first year of operation, Google received over 8700 applications for the 200 funding grants [1]. Due to the overwhelming demand, Google doubled the grants to over 400. Google’s goals were to support Open Source Software projects, give students an opportunity to receive payment for their work and student mentoring from the participating institutions. It is important to remark that Google focuses specially in the mentoring aspects of the students; with the hope that they may become long term contributors to such projects. The task, which lasts for about three months during summer, once completed, would allow for the payment of the student. The participating organizations were also given a comparatively small sum of money for their mentoring of the students. This year, marked the beginning of an ever larger event. Since then, GSoC has grown in popularity and size, it has also refined the way grants are handed to students and the evaluations that need to be done at the end of the task.

## 2 GSoC Today

In order to gain a better understanding on the current size of the GSoC, a summary of the results reported by Google for

the 2021 edition [2] is presented here. There were over 1200 students from 67 countries that successfully completed the accepted tasks within the given time frame. The number of accepted organizations was 199 with over 2100 mentors from 75 countries. Another interesting data point is that 88% of the students answered they would apply to GSoC again.

In late 2021, Google announced major changes for the upcoming GSoC in 2022 [3]. GSoC would no longer be only for students or recent graduates. It will be open to anybody above the age of 18 and would target students, self-taught programmers, returning professionals, etc. On top of this major change, the temporary measure adopted in 2021 due to the pandemic, to allow for longer tasks and more flexibility in the schedule; will become permanent. Therefore, organizations and institutions can propose “medium” size tasks, which should take 175h approximately to complete; or larger projects, taking about 350h to completion.

## 3 An Opportunity for the Ada Community

GSoC presents itself as a great opportunity for Open Source communities to grow and improve their offering, both in size and quality. Language organizations can also apply to GSoC. One example is the participation of the Fortran community in 2021 [4].

The Ada community along with its stakeholders are already involved in several open projects that could benefit from more work. The Ada community could also expand its reach thanks to the paid work offered by GSoC and community/public facing projects. Some ideas that could be proposed as tasks are:

- Alire [5]: Ada package-project manager;
- SPARK: improvements of the tools, packages, update to cover Ada 202X, etc;
- AdaWebPack [6]: run Ada code on the browser using WebAssembly;
- GNAT-LLVM [7]: GNAT frontend using the LLVM compiler infrastructure;
- revamp of the Ada Rapporteur Group (ARG) website;
- revamp of the Ada Reference Manual (ARM) tooling;
- ASIS: improvements and reintegration in upstream GCC;
- AdaControl [8]: update analysis to cover Ada 202X;
- general improvements to libraries, compilers, infrastructure, etc.



## 4 Participating in GSoC

The author would like to disclose that he has no experience participating in GSoC as a student nor as a mentor. The information presented below is a summary of what is published by official Google sources.

The main source of information and news is the official GSoC web-page [9]. It contains news, announcements and links to documentation that is relevant to all participants. The site to gain more information on mentoring can be found in [10], while the participant page is [11]. The timeline for the 2022 edition of GSoC is found in [12].

The final date for mentor applications for the 2022 edition ends on February 21st, at 18:00 UTC. Therefore, it is more than likely that no Ada stakeholder will participate in this edition. However, that is not the goal of this publication. The author would like to bring awareness to the Ada community of the possibilities that GSoC presents and to propose participation in the edition that will take place in 2023.

## 5 Organizing Proposals for the Next Edition

One of the major potential issues in presenting Ada tasks to GSoC would be the organization and mentoring from the different stakeholders. The tools, libraries, etc; present in the Ada ecosystem, are handled by different stakeholders. One such example is the GNAT compiler, which falls within the FSF/GCC organization. Since they have been participating in several editions [13], it may be best for the relevant Ada stakeholders to propose tasks regarding GNAT from within the FSF/GCC organization, instead of separately. However, this may decrease visibility of the participation of Ada in GSoC. Another option is to create a “common front” and propose tasks that may or may not clearly fall within other organizations as an “Ada-Lang” entry. This is the approach that the Fortran community took in 2021. They proposed work on tools and libraries from different authors/groups that did not already have a participating organization behind them. And, as far as the author is aware, Fortran-Lang [4] is a purely community run organization. There are some ISO participating members, but it is not acknowledged as an official Fortran entity.

The efforts to create a common proposal could be handled by the major Ada Open Source developers and communities that promote the use of Ada. This is something that will have to be discussed and readied for the next edition of GSoC if a serious proposal is to be made. And as an obvious reminder, there has to be a will to mentor newcomers; without mentors, the tasks will not be accepted.

## 6 Author’s Transparency

The author would like to disclose that the writing of this article was suggested by Dirk Craeynest (Ada-Europe) in

order to bring awareness of this opportunity to the readers of the AUJ. The author would also like to express his bias towards the libre software community and his will to keep fostering it.

## References

- [1] B. Byfield, “Google’s Summer of Code concludes.” <https://archive.ph/20110521182621/http://www.linux.com/articles/48232#selection-665.0-665.33>, Sep 2005.
- [2] “Google Summer of Code 2021: Results announced!” <https://opensource.googleblog.com/2021/08/google-summer-of-code-2021-results-announced.html>, August 2021.
- [3] “Expanding Google Summer of Code in 2022.” <https://opensource.googleblog.com/2021/11/expanding-google-summer-of-code-in-2022.html>, Nov 2021.
- [4] O. Čertík, M. Curcic, S. Ehlert, L. Kedward, A. Markus, B. Richardson, D. Rouson, and M. Ward, “Fortran-lang accepted to Google Summer of Code 2021.” <https://fortran-lang.org/newsletter/2021/03/09/fortran-lang-accepted-for-google-summer-of-code-2021/>, March 2021.
- [5] A. Mosteo and F. Chouteau, “Alire: Ada Library REpository.” <https://alire.ada.dev/>.
- [6] V. Godunko and M. Reznik, “AdaWebPack.” <https://github.com/godunko/adawebpack/>.
- [7] AdaCore, “GNAT-LLVM.” <https://github.com/AdaCore/gnat-llvm>.
- [8] J.-P. Rosen, “AdaControl.” <https://www.adalog.fr/en/adacontrol.html>.
- [9] Google, “Google Summer of Code.” <https://summerofcode.withgoogle.com/>.
- [10] Google, “What is Google Summer of Code? Mentor page.” <https://google.github.io/gsocguides/mentor/>.
- [11] Google, “What is Google Summer of Code? Participant page.” <https://google.github.io/gsocguides/student/>.
- [12] Google, “Google summer of code 2022 timeline.” <https://developers.google.com/open-source/gsoc/timeline>.
- [13] GCC, “GCC’s GSoC page.” <https://gcc.gnu.org/wiki/SummerOfCode>.

# Tools to get you there. Safely.

Ada and The GNAT Pro High-Integrity Family



[www.adacore.com](http://www.adacore.com)

**AdaCore**  
The GNAT Pro Company

# Ember: An Embedded Robotics Library in SPARK

*Kristoffer Nyborg Gregertsen*

*SINTEF Digital, Trondheim, Norway; email: kristoffer.gregertsen@sintef.no*

## Abstract

*This paper describes the Ember library for high-integrity embedded robotics and GNC applications developed in SPARK 2014 with formal verification. The library is based on generic packages and includes functionality as linear algebra, complex numbers, quaternions, and kinematics. Preliminary test results for linear algebra performance are very promising.*

*Keywords: Numerics, robotics, embedded, SPARK*

## 1 Introduction

Numerical code used for robotics and guidance, navigation and control (GNC) applications have a different set of requirements than general-purpose numerics libraries. The code must run on embedded systems with limited computational resources, and should not include more functionality than is needed. Furthermore, the system interacts with the physical world and failures may cause harm to health and environment. Thus the GNC application often will have real-time, dependability and safety requirements that must be verified. Control engineers typically develop and analyze their algorithms with numerical tools such as MATLAB and Simulink, and could then use a certified auto-coding tool such as QGen to generate code for the target [1]. In other situations it may be desirable to have full control over the code, or more cost-efficient to re-implement it for the target. In both cases it is necessary to have a numerical library for the target that can be verified, for instance using formal verification and SPARK [2].

Ember is a library for embedded robotics developed in SPARK 2014 with the goal to: provide the basic functionality needed for embedded robotics and GNC applications; be optimized for performance on embedded systems; include only the necessary functionality by using Ada generic packages and routines; not have external dependencies other than a minimal run-time environment; and be certifiable to software criticality Level B [3].

Ember has been developed under the I3DS and EROSS projects in the H2020 Strategic Research Cluster (SRC) on Space Robotics. The goal of the SRC is to reduce development time and cost of European space robotics missions by providing a set of common building blocks. In the first call of the SRC, parallel projects developed the common building blocks for RTOS, autonomy, sensor fusion, sensor suites and mechanics, and in the second call these building blocks were used in orbital and planetary tracks. The I3DS project [4] was led by Thales Alenia Space (TAS) and developed a sensor

suite with standardized interfaces. SINTEF and the author led the software architecture and integration work. The EROSS project [5] of the second SRC call is also led by TAS, and develops an architecture for on-orbit servicing of satellites that was successfully demonstrated in the spring of 2021. The newly started EROSS+ project is a continuation of EROSS and will prepare the technology for launch of a demonstrator to space by 2025. SINTEF is responsible for the maturation of the I3DS sensor suite and the overall software architecture in EROSS and EROSS+. One major challenge is to rise the TRL and software maturity to be qualified for software criticality Level B, as most of the software of the first two SRC calls depends on open-source frameworks such as OpenCV and Eigen, and is executed in TASTE [6] and GNU/Linux. The current code-base is mostly in C++ and has not been developed in accordance with software assurance procedures [3] due to limitations in time and budget. On goal for SINTEF in EROSS+ is to replace much of this code-base with software developed using the Ember library and formal verification with SPARK 2014.

## 2 The Ember library

Ember is developed with GNAT Pro and SPARK Pro tools from AdaCore, and is organized as a hierarchy of generic Ada packages. There are two main parent packages as of 2021: `Ember.Numerics` for numerical functionality, and `Ember.Kinematics` for kinematics such as rotation and reference frames used for robotic applications.

### 2.1 Numerics

The numeric package defines a generic package without body for defining mathematical rings as real and complex numbers.

**generic**

```

type T is private;
type Real is digits <>;

with function "+" (Left, Right: T) return T;
with function "-" (Left, Right: T) return T;
with function "*" (Left, Right: T) return T;
with function "/" (Left, Right: T) return T;
with function "--" (Right : T) return T;

with function "abs" (Right : T)
  return Real'Base;
with function "*" (Left : T; Right: Real)
  return T;

Unity : T;
Zero : T;
Commutative : Boolean;

```

**package** Generic\_Ring **is end**;



The linear algebra package supports vector and matrix types and operations on these. The ring package shown above is used as generic parameter so that the same generic packages can be used both for real and complex numbers. In contrast to the standard definition in the Ada numerical annex, these types are of fixed size, and are always indexed from 1.

```
generic
  N : Positive;
  M : Positive;
  with package Field is new Generic_Ring (<>);
  use Field;
package Generic_Matrix is

  subtype Index_1 is Positive range 1 .. N;
  subtype Index_2 is Positive range 1 .. M;

  type Matrix is array (Index_1, Index_2) of T;
  ...
end Generic_Matrix;
```

The vector and matrix packages support operations such as adding and subtracting with the same type, as well as scalar multiplication and division and initialization using the definitions of the generic ring package. For the dot product between matrix and vectors types the generic dot function must be instantiated. The ring of the two input and result types need not be the same, so it is for instance possible to multiply a complex matrix with a real vector.

```
generic
  N, M, L : Positive;
  with package MA is new
    Generic_Matrix (N, L, others => <>);
  with package MB is new
    Generic_Matrix (L, M, others => <>);
  with package MR is new
    Generic_Matrix (N, M, others => <>);
  with function "*"
    (Left : MA.Field.T;
     Right : MB.Field.T) return MR.Field.T;
function MM_Dot
  (A : MA.Matrix;
   B : MB.Matrix) return MR.Matrix;
```

Generic functions are also defined to transpose matrices, to take the power and exponential of square matrices, and special functions for getting the determinant of 2x2 and 3x3 matrix types. LU factorization is used to solve NxN linear systems and get their determinant and inverse.

A package with elementary functions is defined in the same way as for the Ada numerical annex, but with pre-and post-conditions, for instance demanding that the input to the Log function must be larger than 0, and guaranteeing that the result of the exponential function is always larger than 0.

```
function Log (X, Base : Real) return Real with
  Pre => (X > 0.0 and Base > 0.0 and Base /= 1.0);

function Exp (X : Real) return Real with
  Post => (Exp'Result > 0.0);
```

As of now the elementary functions are implemented with the standard Ada library, but they will be rewritten from scratch as Ember is supposed to have minimal dependencies.

Complex numbers are defined as in the Ada numerical annex, but pre- and post-conditions are used, and pure imaginary

numbers are not defined as a separate type. Quaternions are similar to complex numbers but are non-commutative, and consist of four scalars represented in Ember as a real scalar part and an imaginary part with components (i, j, k).

## 2.2 Kinematics

The kinematics package defines types and operations for rotation and translation in 3D space. The rotation matrix with  $\det R = 1$  is used for rotation of vectors, either around an axis, as Euler parameters (rotation around vector), or roll-pitch-yaw (RPY). Similarly, unit quaternions are used for rotation either as Euler parameters or RPY. The 6 degrees-of-freedom (DOF) poses are described as reference frames with an offset and rotation matrix, and allow vectors in one frame to be translated to and from other reference frames. Finally, the screw type can be used for calculations on pairs of vectors with linear and rotational components.

The reference frame tree package allows the entities of a system to be related to a global reference frame and each other through one or more intermediate steps. The tree structure is enforced by pre-conditions saying that a frame can only have a parent frame with ID lower than its own (with 0 being the global frame). This ensures that circular references cannot occur. When computing the pose of one frame as seen from another the shortest path in the tree is taken.

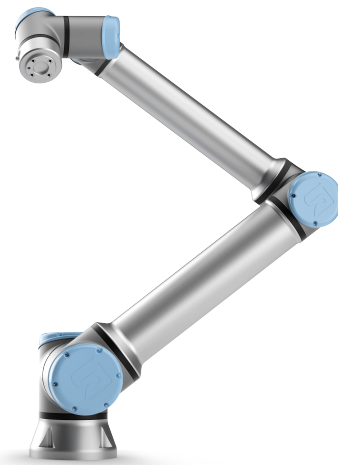


Figure 1: The UR-10 manipulator from Universal Robotics.

The manipulator package represents a robot arm with linked segments that can be rotated or translated relative to its parent. This allows to represent for instance a 6-DOF robot arm as the UR-10 manipulator shown in Figure 1, and calculate the pose of the end-effector given the joint-configuration. The manipulator consists of a fixed number of segments given as generic parameter along with a frame tree package. It is initialized with a base ID that may or may not be the global frame. Each segment can then be configured with a segment type (fixed, rotational or translational), and a frame (offset and rotation) for the tip of the segment relative to the joint. The tip of a segment is the reference for the next segment in the robot, and so on until the end-effector. When the joint-configuration of the manipulator is set the corresponding reference frames of the tree are updated accordingly. In this

way all segments can be related to other reference frames of the tree, for instance to calculate the pose of the end-effector relative to some object that is going to be gripped. The inverse kinematics operation of calculating the needed joint-configuration for a desired end-effector pose is not yet implemented.

When calculating motion (or guidance) of robotic manipulators it is important to check for collisions and that safety zones are not violated. To support this, Ember has a package for basic 3D geometric shapes as capsules, spheres, cuboids, and cones, each with physical dimensions and linked to a reference frame in the provided tree. The package supports calculating the internal distance to find how far an object is from going out of a safety zone, and the external distance to find how far two objects are from colliding. This package can be used to check if a given joint-configuration of a manipulator would result in collisions, either between segments or other obstacles, and if any segment would go outside safety zones. A manipulator such as the UR-10 shown in Figure 1 could be represented with one or two capsules for each segment, depending on the desired tolerances. A safety-zone will typically be statically bound to the base of the manipulator, but could also be dynamic.

### 3 Performance test

A simple performance test is run to get some preliminary results of the Ember library compared to the Ada numerics package and the C++ Eigen library. The test runs a simple linear system update  $x \leftarrow A \cdot x + b$  for  $x \in \mathbb{R}^3$  with 9 multiplications and 9 additions. The total execution-time of 100 million such updates of  $x$  is measured 20 times.

For Ember, one must define the real domain represented with IEEE 754 double-precision floating-point type, the 3-dimensional vector and 3x3 matrix, and the operator to multiply the matrix with the vector:

```

procedure Ember_Perf is

  package RT is new Real_Types (Float_64);

  package RV3 is new
    Linear_Algebra.Generic_Vector
      (3, RT.Real_Field);
  package RM3 is new
    Linear_Algebra.Generic_Matrix
      (3, 3, RT.Real_Field);

  use type RV3.Vector;
  use type RM3.Matrix;

  function "*" is new
    Linear_Algebra.MV_Dot
      (3, 3, RM3, RV3, RV3, "*");

  A : constant RM3.Matrix :=
    ((0.1, 0.2, 0.3),
     (0.2, 0.1, 0.3),
     (0.3, 0.2, 0.1));

  B : constant RV3.Vector :=
    (1.0, 2.0, 3.0);

  X : RV3.Vector := RV3.Zeros;

```

```

T0, T1 : CPU_Time;

begin
  for I in 1 .. 20 loop
    T0 := Clock;

    for J in 1 .. 100_000_000 loop
      X := A * X + B;
    end loop;

    T1 := Clock;
    Put (To_Duration (T1 - T0), 0);
    New_Line;
  end loop;
  ...
end Ember_Perf;

```

Not shown in the listings is the with/use statements and the final output of  $x$  to validate and enforce its calculation. The test using the Ada Numerics package is exactly the same except for the declarations of the matrix and vectors.

The C++ test using Eigen is also similar in structure:

```

int main()
{
  Eigen::Matrix<double, 3, 3> A;

  A << 0.1, 0.2, 0.3,
      0.2, 0.1, 0.3,
      0.3, 0.2, 0.1;

  Eigen::Vector3d b(1.0, 2.0, 3.0);
  Eigen::Vector3d x(0.0, 0.0, 0.0);

  long t0, t1;

  for (int i = 0; i < 20; i++) {
    t0 = clock();

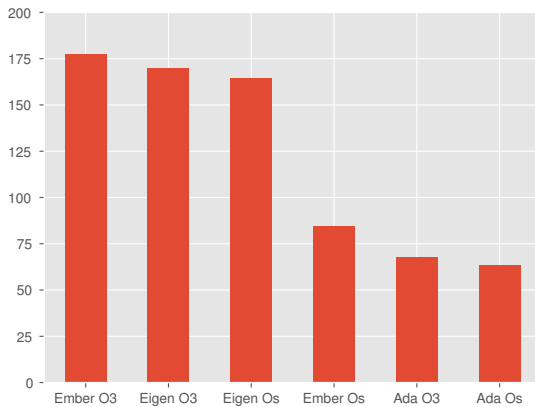
    for (int j = 0; j < 100000000; j++) {
      x = A * x + b;
    }

    t1 = clock();
    std::cout << 1e-9 * (t1 - t0) << std::endl;
  }
  ...
  return 0;
}

```

The three test programs were compiled and linked using GNAT Pro 21 with GCC optimization flags “-O3” for speed and “-Os” for minimal code size, and was run on a Core i7-6700 @ 3.4 GHz with Arch Linux operating system.

Figure 2 shows the results in millions of iterations per second for the median of the six test runs. To remove outliers caused by interference the median of the measurements was used. In general there was very low variance for the 20 iterations. As it can be seen Ember and Eigen have similar measurements when optimized for performance, while Ember has lower performance when optimized for space. The Ada numerical library has lower performance than Ember and Eigen in both cases, and is not much affected by optimization flags. Notice that the Ada test programs use the execution-time while the C++ test use the high-resolution clock, the test were also run with the real-time clock for Ada with similar results.



**Figure 2: Preliminary performance comparison in millions of iterations per second with “O3” and “Os” optimization flags.**

## 4 Discussion

The use of SPARK 2014 has enforced a disciplined style of coding, and allowed to catch basic errors such as for data flow making sure that all variables are initialized prior to usage. This is referred to as the bronze level by AdaCore. Also, the code is found to be free of run-time exceptions other than floating-point overflow. This is referred to as silver level. However, going to the full gold level with full formal verification is not straightforward for numerical algorithms, as the prover has limitations in understanding of numerical operations. While rare when working with physical units in GNC applications, floating-point overflow could arise from numerical operations, for instance when dividing by a very small number, or adding two very large numbers. Further work is needed to define lemmas to aid the prover, and to make restrictions on inputs to prove absence of overflows.

Still there are many algorithms that are hard to prove formally, such as matrix inversion, and testing is needed both to prove correct functionality and to evaluate the numerical stability. The AUnit framework is used for setting up and executing test harnesses of the different functionality. Test code is only written for non-trivial algorithms that cannot easily be proven. For instance, the inverse is found for a set of random matrices, and it is shown that the product of the matrix and its inverse is sufficiently close to the identity matrix. It would be desirable to automatically create tests code from pragmas specifying properties of types and functions, such as the relation between a matrix and its inverse.

The design choice of using generic packages is motivated by a desire to only include the functionality that is needed in the final build. However, the need to instantiate generic packages for different sized vectors and matrix, and generic functions for each dot product operation used seems to go against this goal. The rationale is that robotics and GNC applications often will use only a limited set of rather small vector and matrix sizes, and that the benefits in execution-time and possibility of in-lining simple operations will justify this. This approach is also used by the Eigen framework that

has inspired the linear algebra parts of Ember. As shown in preliminary test results in 2, the Ember library matches the performance of Eigen on a Core i7 when in-lining is used, and is able to perform more than double the number of iterations per second compared to the identical code using the Ada numerical annex. The performance gains are reduced when in-lining is turned off by optimizing for space. More work is needed to test for a larger and more realistic application on an embedded target and to compare the size of the binaries.

## 5 Conclusion and further work

While the numerical functionality of Ember has been stable for more than one year, the kinematics packages are still being developed and tested. Also, there is still much work needed to provide the all the functionality needed for robotics applications. Planned features include support for inverse kinematics, (Extended) Kalman filters used for estimation and navigation, and basic image processing like the core functionality of OpenCV. However, it is important to notice that Ember is not intended to replace general-purpose numerical tools like MATLAB and the Python NumPy/SciPy packages. Only the functionality needed at run-time to and that can be implemented with reasonable computational resources in a high-performance embedded system is to be provided.

Ember has not yet been released to the public under an open-source license, the release is planned in the end of 2021. It is hoped that the Ember library can contribute to increased use of Ada and SPARK 2014 in the robotics and control field, and result in embedded software that is suitable for high-integrity applications and easier to verify.

## Acknowledgements

Many thanks to AdaCore that has provided both the needed tools and consultation free of charge. This work has been funded under the H2020 research and innovation programme under the I3DS grant agreement No 730118, and under the EROSS grant agreement No 821904.

## References

- [1] T. Naks, M. A. Aiello, and S. T. Taft, “Using SPARK to ensure system to software integrity: A case study,” *Ada User Journal*, vol. 40, no. 4, pp. 226–229, 2019.
- [2] P. Neto, J. Tojal, J. Veríssimo, and S. M. de Sousa, “Towards a formally verified space mission software using SPARK,” *Ada User Journal*, vol. 40, no. 4, pp. 243–246, 2019.
- [3] ECSS, “ECSS-Q-ST-80C Rev. 1: Space product assurance - software product assurance,” 2017.
- [4] V. Dubanchet and S. Andiappane, “Development of I3DS: An integrated sensors suite for orbital rendezvous and planetary exploration,” in *i-SAIRAS 2018*.
- [5] V. Dubanchet, J. B. Romero, K. N. Gregertsen, *et al.*, “EROSS project - European autonomous robotic vehicle for on-orbit servicing,” in *i-SAIRAS 2020*.
- [6] M. Perrotin, E. Conquet, J. Delange, *et al.*, “TASTE: A real-time software engineering tool-chain overview, status, and future,” *Lecture Notes in Computer Science*, vol. 7083 LNCS, pp. 26–37, 2011.

# Queuing Ports for Mesh Based Many-Core Processors

*David García Villaescusa, Mario Aldea Rivas, Michael González Harbour*

*University of Cantabria, Avenida de los Castros S/N, Santander; email: {garciavd, aldeam, mgh}@unican.es*

## Abstract

*This paper presents the implementation of Queuing Ports, a blocking communication protocol developed for many-core architectures that perform a synchronized communication between cores without the need of polling. This implementation has been performed on M2OS-mc, a Real-Time Operating System (RTOS) that has already been tested in the Epiphany processor. The extension presented is based on the ARINC-653's Queuing Port communication primitive and gives an alternative to the implementation based in the ARINC-653's Sampling Port communication primitive previously developed.<sup>1</sup>*

*Keywords: many-core, queuing-port, epiphany, task synchronization, M2OS.*

## 1 Introduction

Processor manufacturers are always looking for ways to increase the capabilities of their products. The performance improvement of the processor technology is currently focused on integrating more cores in the same System-on-Chip (SoC), allowing us to parallelize execution.

The main problem a real-time system has to face in a current COTS processor is the existence of a shared bus as the way to communicate the cores among them and with the shared memory. Fortunately, the increasing number of cores in the same SoC is driving the processor manufacturers to more scalable solutions avoiding that shared bus.

One of this solutions is using a 2D mesh Network-on-Chip (NoC) like the one present in the Epiphany [1] many-core processor, which has 16 cores connected by a NoC consisting of a 4x4 2D mesh as shown in Figure 1. This is the processor where the developments described at this paper have been tested.

The existence of suitable hardware is not enough to implement a real-time system. Adequate software platforms are also necessary to have a fully capable real-time system. M2OS-mc [2] is the Real-Time Operating System (RTOS) picked as a starting point to develop the Queuing Port communication mechanism based on the existing Garrido's ARINC-653 implementation [3].

<sup>1</sup>This work has been financed by the Graduate Grant Program of the University of Cantabria and by the Spanish Government and FEDER funds (AEI/FEDER, UE) under Grant TIN2017-86520-C3-3-R(PRECON-I4)

M2OS-mc4, as well as M2OS, is written in Ada as this language has specialized features supporting low-level real-time, safety-critical and embedded systems programming. M2OS alongside M2OS-mc are available on-line at the website <sup>2</sup>, and are distributed under a GPLv3+ license.

A typical Ada application executed on M2OS-mc in the Epiphany processor is composed of several tasks running in the different cores, with one or more tasks in each core. Tasks in the same core are executed under the one-shot non-preemptive scheduling policy implemented by M2OS. The messages between tasks allocated in different cores will travel through the NoC.

The paper continues by exposing related work in Section 2. In Section 3 the Epiphany processor is exposed. Section 4 refers to the work done in the RTOS used, the M2OS. Section 5 describes the new communication primitive implemented: the Queuing Ports. Finally, Section 7 shows the paper conclusions and future work.

## 2 Related work

M2OS-mc was initially implemented using Sampling Ports [2] as the communication mechanism. That implementation gives the RTOS a way to communicate in a synchronized fashion through the different cores of the many-core processor with a bounded temporal behavior. The temporal impact of the RTOS has been studied and modeled using the MAST [4] tool.

While the Sampling Ports available in M2OS-mc there are no blocking operations and, therefore, when a reading task is waiting for a message it has to periodically poll for its arrival. This active wait introduces timing overhead to other tasks. Another possible downside of the Sampling Ports is that a writing process will overwrite any not already read message waiting at the Sampling Port and some messages could be missed.

To avoid those downsides, a blocking messaging mechanism has been implemented based on the ARINC-653 Queuing Port.

Garrido [3] presented an implementation for the ARINC-653 synchronized communication ports. The Sampling Port synchronization mechanism developed for M2OS is based on those implementations as well as the Queuing Port mechanism presented in this paper.

<sup>2</sup><https://m2os.unican.es>



### 3 Epiphany

The Epiphany processor is integrated in the Parallella [8] development board. It is a processor with 16 cores connected through a 2D 4x4 mesh NoC as seen in Figure 1. Each core of the Epiphany is an eCore, whose architecture is designed by Adapteva. The eCore executes its instructions in order, with a frequency of 600 MHz. It consists of an integer ALU, floating-point unit, a debug unit, an interrupt controller, a general purpose program sequencer and a 64-word general purpose register file. Each core has 32KB of local memory. The architecture is supported by the GCC compiler and has libraries for OpenMP and MPI.

Any eCore can access the local memory of the rest of the eCores using a range of special global addresses. An eCore's local memory can be written and read without any hardware limitations, but the memory size. Writing to other eCore memory generates a message that must travel through the NoC. Reading from other eCore memory generates a reading request that must travel through the NoC and once it reaches its destination it will generate a writing message to the eCore sending the read request, with the requested data.

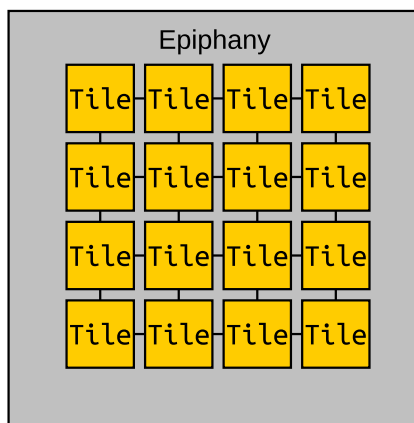


Figure 1: Epiphany many-core topology

### 4 M2OS

M2OS [9] [10] is a small RTOS that allows running multitasking applications in small microcontrollers with scarce memory resources.

M2OS implements a simple scheduling policy based on non-preemptive one-shot tasks, which requires a very small memory footprint. This policy allows the same stack area to be shared by all the tasks and, consequently, the system only needs to allocate a stack area large enough to fit the largest task stack.

One-shot tasks do not keep any local state in the stack between releases. They are made of initialization instructions executed only at the first release of the task and job instructions executed at subsequent releases. The job instructions must end in a blocking operation to wait for the next release event.

Since the scheduling policy is non-preemptive, a running task will not release the CPU until it reaches a blocking operation.

M2OS is written in Ada and it is the base of a simplified Run-Time System for the GNAT Ada compiler. This RTOS has been developed for Arduino Uno and STM32F4. It is intended to be easily ported to different platforms. All the hardware dependent part is encapsulated in a Hardware Abstraction Layer (HAL), which is the only code that has to be modified to port the kernel to a new platform.

#### 4.1 M2OS-mc

The many-core version of M2OS (called M2OS-mc [2]) has a microkernel implementation which means that independent instances of the RTOS are located at each core. The kernel of each core will schedule the execution of all the tasks allocated there. M2OS-mc provides two mechanisms to communicate tasks in different cores: the Sampling Ports (presented in [2]) and the Queuing Ports (described in this paper).

### 5 Queuing Ports

A Queuing Port consists in a circular FIFO queue. A reader is blocked when no data is stored at the accessed Queuing Port and it is activated when new data arrives.

A fixed number of Queuing Ports are assigned per core. These Queuing Ports are identified by an index. Reader and writer tasks must know the index and the host core of the Queuing Port they are going to use to share the data.

Queuing Ports are implemented as an Ada record with the following fields:

- **Initialized.** A boolean to make sure the Queuing Port has been initialized before using it.
- **Mutex.** The spinlock to serialize accesses to the Queuing Port.
- **Queue\_Size.** The maximum number of elements the Queuing Port is able to store.
- **Tail.** The index of the most recent element written to the Queuing Port.
- **Head.** The index of the first element to be read from the Queuing Port.
- **N\_Elements.** The number of elements currently stored at the Queuing Port.
- **Element\_Size.** The size of each element stored at the Queuing Port. Every element has the same size.
- **Elements.** The pointer to the area where the elements of the Queuing Port are being stored. The data structure is an array of N\_Elements of size Element\_Size. This array must be created by the task initializing the Queuing Port.
- **Core.** The core where the Queuing Port is located.

The usage of the Queuing Port is illustrated in Figure 2 where two Queuing Ports can be seen, only one of them initialized. It can be seen that a memory area is located at the local memory of the 0x0 core pointed by the Elements field.

The interface to manage the Queuing Port is show in Listing 1. The parameters Core and Index of the procedures in the interface allow identifying the Queuing Port.

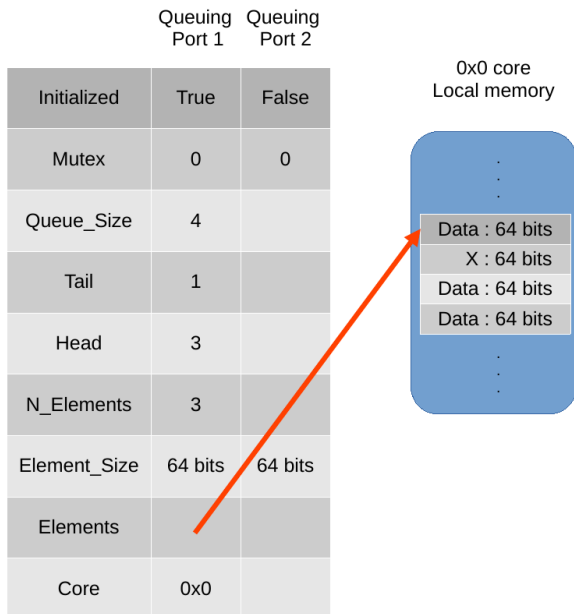


Figure 2: Queuing Port use example

```

function Init_Queueing_Port (Index : in QP_Index;
Size_Q : in Unsigned_32;
Data_Address : in System.Address;
Data_Size : in Unsigned_32;
Core : in E_Lib.Core)
return QP_Access;

procedure Write_Queueing_Port (QP : in out QP_Access;
Orig : in System.Address;
Orig_Size : in Unsigned_32;
Successful : out Boolean);

procedure Read_Queueing_Port (QP : in out QP_Access;
Dest : in System.Address;
Dest_Size : in Unsigned_32;
Successful : out Boolean);

```

Listing 1: Queuing Port interface

The **Init\_Queueing\_Port** procedure initializes the Queuing Port. This procedure must be executed before any other operation. Before initializing a Queuing Port the task calling the procedure must allocate the necessary data structure (an array) to store all the data. The procedure must receive the size of a single data element (every data has the same size), the maximum number of elements and the address of the data structure provided to store the messages.

The **Write\_Queueing\_Port** procedure receives the address of the data to be written and its size. The size must be the same as the Element Size of the Queuing Port. It has an output boolean parameter indicating if the write operation into the Queuing Port that has index N has been successful or the data could not be written because the Queuing Port is already full.

The **Read\_Queueing\_Port** procedure tries to read from the Queuing Port. If the Queuing Port has no data, the calling task blocks. This procedure receives the address where the read data must be stored and the size of the variable created

for that purpose. It indicates the success of the procedure with the out parameter Successful.

The implementation of a Read Task in M2OS-mc must take into account that M2OS-mc implements a non-preemptive scheduling policy, which implies that trying to read from an empty Queuing Port will cause the calling task to finish its current activation and block. For that reason, the data is used in the initial part of the body, previous to the read operation. A read operation is also included in the initialization procedure of the task. The Read Task is shown in Listing 2 alongside with the Write Task, that must wait for the Queuing Port to be initialized. The schematic shown in Listing 2 has been done considering the Read Task to be hosting the data and the queue, which is the most efficient solution.

```

procedure Read_Task_Init is
-- Initialize the task
QP := Init_Queueing_Port
(Port Index, Size_Q, Data_Q'Address, Unsigned_64'Size, Core);

Read_Queueing_Port
(QP, Var'Address, Var'Size, Success);
end Read_Task_Init;

procedure Read_Task_Body is
begin
-- Use data

Read_Queueing_Port
(QP, Var'Address, Var'Size, Success);
end Read_Task_Body;

procedure Write_Task_Init is
-- Initialize the task

QP := Get_Queueing_Port
(Target_Core, Port Index);
while (Check_Queueing_Port_Initialized (QP) = False) loop
null;
end loop;
end Write_Task_Init

procedure Write_Task_Body is
begin
-- Generate data
Write_Queueing_Port (QP, Data'Address, Data'Size, Success);
-- Task work post write
-- Task Delay
end Write_Task_Body;

```

Listing 2: Read and Write Tasks implementation

## 6 Implementation in M2OS-mc

In order to allow the RTOS to have the Queuing Ports implemented some additions have been done.

A priority-sorted queue has been created (**QP\_Blocked\_Task\_Queue**) to store the tasks while they are blocked waiting for new data to be written to a Queuing Port.

The Task Control Block (TCB) has two new fields shown in Listing 3. One that points to the Queuing Port that blocked the task (**QP\_Where\_Blocked**) and another one that indicates

where the task wants to store the data read from the Queuing Port. The data will be stored there once the dispatcher awakens the task.

```
QP_Where_Blocked : Queueing_Port'Access;
QP_Data : QP_Data_Type;
```

### Listing 3: Task record addition

In order to implement the Queuing Ports it has been necessary to modify the dispatching algorithm of M2OS-mc. The developed implementation consists of trying to find a blocked task that has any pending message with a higher priority than the task selected from the Ready Queue, which is the task with the highest priority of that queue. The new part of the Dispatcher algorithm is shown in Listing 4.

The `Pop_QP` procedure takes the first message of the Queuing Port and copies it to the Data, both passed as parameters.

```
Head_Ac := Ready_Queue.Head;
Exit_BQ := False;
Th_Ac := QP_Blocked_Tasks_Queue.Head;
while Th_Ac /= Null_TCB_Ac and not Exit_BQ loop

  if Head_Ac /= TCBS.Null_TCB_Ac
  and then Th_Ac.Priority > Head_Ac.Priority then
    exit; -- Exit when there is no higher priority task
    -- in QP_Blocked_Tasks_Queue
  end if;

  Th_Ac_Prev := null;

  E_Lib.E_Mutex_Lock (Th_Ac.QP_Where_Blocked.Core.Row,
                    Th_Ac.QP_Where_Blocked.Core.Col,
                    Th_Ac.QP_Where_Blocked.Mutex);

  if Th_Ac.QP_Where_Blocked.N_Elements > 0 then
    Pop_QP (Th_Ac.QP_Where_Blocked, Th_Ac.all.QP_Data);

    -- To ready Queue
    Th_Ac.all.Status := Ready;
    RQ.Enqueue_Ready_Thread (Th_Ac);

    -- Enqueuing will make it exist in the next
    -- iteration of the loop
    Head_Ac := RQ.Head_Of_Ready_Queue;
    -- Remove from blocked tasks
    if Th_Ac = Head_Of_QP_Blocked_Tasks_Queue then
      Dequeue_Head_Of_QP_Blocked_Tasks_Queue;
    else
      if Th_Ac_Prev /= null then
        Th_Ac_Prev.Next := Th_Ac.Next;
      end if;
    end if;

    Exit_BQ := True;
  end if;

  E_Lib.E_Mutex_Unlock (Th_Ac.QP_Where_Blocked.Core.Row,
                      Th_Ac.QP_Where_Blocked.Core.Col,
                      Th_Ac.QP_Where_Blocked.Mutex);

  Th_Ac_Prev := Th_Ac;
  Th_Ac := Th_Ac.Next;
end loop;

-- From here, activate the first Ready_Queue
```

### Listing 4: Dispatching blocked task

## 7 Conclusions and future work

The ARINC's 653 Queuing Port has been implemented for M2OS-mc and it has been tested with the Epiphany processor.

This implementation gives M2OS-mc an alternative to Sampling Ports for having synchronized communication among cores.

As future work, we have to perform more tests to merge the Queuing Ports into the M2OS-mc project. Another step is to study the temporal behavior of the Queuing Port implementation for different sizes and distances to find the way to model its behavior in a way that allow us to analyze a system using MAST [4], as it was previously done for the Sampling Port implementation [2].

## References

- [1] A. Olofsson, T. Nordström, and Z. Ul-Abdin, "Kick-starting high-performance energy-efficient manycore architectures with epiphany," 2014.
- [2] D. García Villaescusa, M. Aldea Rivas, and M. González Harbour, "M2OS-Mc: An RTOS for Many-Core Processors," in *Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021)* (M. Bertogna and F. Terraneo, eds.), vol. 87 of *OpenAccess Series in Informatics (OASICs)*, (Dagstuhl, Germany), pp. 5:1–5:13, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.
- [3] J. G. Balaguer, J. R. Z. Flores, and J. A. de la Puente Alfaro, "Arinc-653 inter-partition communications and the ravenscar profile," *Ada Letters*, vol. 35, no. 1, pp. 38–45, 2015.
- [4] M. González Harbour, J. L. Medina, J. J. Gutiérrez, J. Palencia, and J. Drake, "Mast: An open environment for modeling, analysis, and design of real-time systems," *1st CARTS Workshop*, January 2002.
- [5] E. Enterprise, "Erika3." [Online; accessed 29-January-2020].
- [6] eSol, "Scalable and High-performance Real-Time OS available for various types of processors." [Online; accessed 29-January-2020].
- [7] H. Almatary, *Operating System Kernels on Multi-core Architectures*. PhD thesis, University of York, January 2016.
- [8] A. Olofsson, T. Nordström, and Z. Ul-Abdin, "Kick-starting high-performance energy-efficient manycore architectures with epiphany," 2014.
- [9] M. Aldea Rivas and H. Pérez Tijero, "Leveraging real-time and multitasking Ada capabilities to small microcontrollers," *Journal of Systems Architecture*, vol. 94, pp. 32 – 41, 2019.
- [10] M. Aldea Rivas and H. Pérez Tijero, "Proposal for a new Ada profile for small microcontrollers," *Ada Lett.*, vol. 38, p. 34–39, July 2018.

# More Ada in Non-Ada Systems

*A. Marriott, U. Maurer*

*White Elephant GmbH, Beckengässchen 1, 8200 Schaffhausen, Switzerland; email: software@white-elephant.ch*

## Abstract

*This article is based on the industrial presentation “More Ada in non-Ada systems” which was given at the 2021 Ada-Europe virtual conference.*

*The presentation was an update to the presentation given at the Ada-Europe 2018 conference in Lisbon entitled “Using Ada in non-Ada systems” and that was subsequently published in the Ada User Journal [1].*

*The work used GCC 7.4.1 built by AdaCore and made available as part of their 2019 Windows hosted GNAT for ARM GPL community distribution. The GNAT sources referred to in the presentation were also obtained from this distribution.*

*Keywords: GCC, Modula-2, C, ZFA, ARM*

## 1 Previously

If we restrict ourselves to a subset of Ada, the so called Zero Footprint Ada (ZFA), then it is possible to write applications that use a mixture of Ada and C. The Ada we use is not the Ada with which we are most familiar, it is a very cutback version – but even so, despite all the restrictions, it can still be beneficial to write code in Ada rather than in C.

We believe that there are many reasons why one should program in Ada rather than C. However, from a business perspective, these reasons may be insufficient to warrant the complete rewrite of a large, stable and successful system.

Rather than trying to advocate that our systems should be rewritten completely in Ada we showed how existing code could be supplemented by code written in Ada.

In 2018 our work was a proof of concept. At that time our management had no intention nor desire to write any code in Ada. The existing software and expertise were in Modula-2, a language that can be considered to be a simpler yet similar language, at least in comparison with alternatives such as the dreaded C.

## 2 Data compression

However, all this changed when it was decided that downloads to our ARM based processors should be compressed. As products develop, they tend to grow in both complexity and size, yet the speed of the field bus used to transfer the executable images remains the same. This results in the time to download getting longer and longer.

As a consequence, it was decided that it would be an interesting salable feature if this time could be reduced by compressing the download. Because we lack the required expertise to develop this sort of software, this decision would

require us to use some sort of library - either binary, or just as bad, source written in C.

Another alternative would be to use the ZipAda [2] open-source code written by our colleague Gautier de Montmollin. But, as the library’s name suggests, this software is written in Ada. This presented us with the choice of either using it directly in Ada or converting it into Modula-2, which, as I’ve already alluded, is a very similar language. Similar in theory, but in practice it rather depends on what features of Ada one has used. Unfortunately for us, Gautier had used many features of Ada that aren’t supported by Modula-2 - thereby making a translation virtually impossible. Or at least not without the risk of introducing all manner of interesting and hard to find bugs.

The solution was to use the proof of concept work we had made in 2018 and use his source code directly. I think it says a lot about the portability of software written in Ada that we could use the ZipAda software with only the most minor of changes.

The software was designed to run on a PC reading data from a file and compressing it to make what is generally referred to as a Zip File. Instead, we use a PC to read the executable to be downloaded, compress this and then transmit the compressed byte stream to the microprocessor which then uses the same software to decompress the byte stream back into the executable.

Amazingly this all worked, with almost no effort. We took software written in Ada that was designed and primarily works on PCs, recompiled it using an Ada compiler for ARM and then linked this into our product – that was otherwise written entirely in machine generated C.

This then was the first commercial use of the work I presented in 2018 as a theoretical exercise.

## 3 Floating-point

Up until now all the microprocessors that we have used have been without floating-point units.

Because floating-point without hardware support is incredibly inefficient, we explicitly prohibited its use by not recognizing the Modula-2 floating-point syntax.

In the Ada work I presented in 2018 we used the pragma Restrictions (No Floating Point) in System.ads to impose the same restriction on routines written in Ada.

However, our latest target microprocessor has a floating-point unit and so, unsurprisingly, there are plans to use it, so the dilemma has been to decide how to use it.

If we permit the floating-point syntax in Modula then we may encounter problems with our Modula to C translator – which, obviously, has never been used to translate anything concerning floating-point. Then there is the problem of what exactly is floating-point in C and then, lastly, what libraries are available, how good are they and can we obtain the sources for them?

An alternative strategy, and what we have ultimately decided to do, is to forget Modula-2 and, because only our ARM microprocessor has an FPU, write all our floating-point code in Ada.

Allowing the Ada syntax was easy – all we had to do was remove the pragma Restrictions (No\_Floating\_Point). The Ada compiler can be directed to generate code using the ARM floating-point instructions by including the switch “-mfloat-abi=hard” and the switch “-mfpu” to describe what sort of FPU the processor has.

Therefore it is relatively easy to support code that merely performs simple arithmetic.

The fun starts when one wants to use functions that are provided by the package Ada.Numerics.Elementary\_Functions and/or its long float counterpart Ada.Numerics.Long\_Elementary\_Functions.

Trying to compile the Ada statement

```
X:= Ada.Numerics.Elementary_Functions.Tan(45.0);
```

will produce the error message

```
"Ada.Numerics" is not a predefined library unit
```

This means that GNAT cannot find the specification for Ada.Numerics in the source file search path. This specification file must be provided even though it only contains two constants: Pi and e.

Moreover, GNAT requires that the specification is provided in a file with the crunched file name a-neri.ads.

Once this file is made available, GNAT will then require the file a-nuelfu.ads and once this is provided, it will start requiring various system definitions, starting with System.Generic\_C\_Math\_Interface contained in

```
s-gcmain.ads
```

To cut a long story short, to use Ada’s floating point elementary functions requires that the GNAT compiler be able to find 14 specification packages in files with crunched filenames.

Helpfully the compiler tells you what is missing and therefore these files are relatively easy to find and place in the source path.

The 14 files are:

- a-neri.ads
- a-ndelfu.ads
- a-nuelfu.ads
- a-nlelfu.ads
- s-gcmain.ads

- s-libsin.ads
- s-libpre.ads
- s-libdou.ads
- s-exnllf.ads
- s-fatflt.ads
- s-fatlfl.ads
- s-fatgen.ads
- s-fatgen.adb
- s-lisisq.ads

Once GNAT can find these files it will compile. However, it will then fail at link time.

In my tangent example it will fail with

```
undefined reference to
```

```
`ada__numerics__elementary_functions__tan'
```

because although we have provided the specification of the tangent function, we haven’t provided the implementation.

Unfortunately, if we take the source of the implementation and try to compile a-nuelfu.adb, GNAT produces the error message

```
user-defined descendants of package Ada are not allowed
```

The work-around to this is to rename the sources and to export the functions with the name they would have had, had they been compiled as descendants of Ada. If we place the objects into a static library (object archive) and provide this library explicitly in the linker script, the GCC can resolve the references, and all is well.

Our Ada is used in conjunction with C machine generated from source written in Modula-2. We called this mixture of Ada and Modula, Adam. For this reason, we chose to base our library on the root package Adam.

For example, for the Tangent function we copied the source file a-nuelfu.ads and renamed it to Adam-Numerics-Elementary\_Functions.ads.

We then modified the contents to rename the package name to Adam.Numerics.Elementary\_Functions and exported every function with the External name prefixed by "ada\_\_numerics\_\_elementary\_functions\_\_"

```
package Adam.Numerics.Elementary_Functions with  
Preelaborate is
```

```
Prefix : constant String :=
```

```
"ada__numerics__elementary_functions__";
```

```
function Tan (X : Float) return Float
```

```
with Inline,
```

```
Export,
```

```
External_Name => Prefix & "tan";
```

```
end Adam.Numerics.Elementary_Functions;
```

For the implementation we renamed the file a-nuelfi.adb to Adam-Numerics-Elementary\_Functions.adb and modified

the contents so that the package name became `Adam.Numerics.Elementary_Functions`.

We had to do this for the following 21 files:

- `Adam-Numerics.ads`
- `Adam-Numerics-Elementary_Functions.ads`
- `Adam-Numerics-Elementary_Functions.adb`
- `(a-nuelfu.adb)`
- `Adam-Numerics-Long_Elementary_Functions.ads`
- `Adam-Numerics-Long_Elementary_Functions.adb`  
`(a-nlelfu.adb)`
- `Adam-Generic_C_Math_Interface.ads`
- `Adam-Generic_C_Math_Interface.adb`
- `(s-gcmain.adb)`
- `Adam-Libm.ads`
- `Adam-Libm.adb (s-libm.adb)`
- `Adam-Libm-Single.ads`
- `Adam-Libm-Single.adb (s-lisinsq.adb, s-lisisq.adb)`
- `Adam-Libm-Double.ads`
- `Adam-Libm-Double.adb (s-libdou.adb)`
- `Adam-Exn_LLF.ads`
- `Adam-Exn_LLF.adb (s-exnllf.adb)`
- `Adam-Generic_Attributes.ads`
- `Adam-Generic_Attributes.adb (s-fatgen.adb)`
- `Adam-Attributes.ads`
- `Adam-Attributes.adb (s-fatflt.adb)`
- `Adam-Long_Attributes.ads`
- `Adam-Long_Attributes.adb (s-fatlfl.adb)`

Other than renaming we did not have to modify the code very much, but some minor changes had to be made:

### 3.1 Exception handling

In ZFA, exceptions cannot be propagated outside the procedure, so either they must be caught and handled (usually by an `SVC` instruction) or suppressed if the exception is thought to be impossible to raise. I refer you to my previous presentation [1] for a description on how to do this and why it has to be done.

### 3.2 Square-Root

In `Adam-Libm-Single.adb` and `Adam-Libm-Double.adb` we implemented the square root function as the built-in intrinsic so that the FPU `VSQRT` instruction is used rather than the acres of code provided in `s-lisinsq.adb` and `s-lidosq.adb` respectively.

### 3.3 Copy\_Sign

The function `Copy_Sign` in `s-fatgen.adb` didn't compile using GCC v7.4.1.

GNAT issued the error message "Incorrect context for intrinsic convention" on the pragma import for the function `Is_Negative`.

Our solution was simply to replace this call with a comparison with 0.0 and the presentation described the code this produced and questioned whether or not it was actually correct.

All this was rendered moot by AdaCore releasing GPL 2021

In this release GNAT (GCC v10.3) no longer recognizes the intrinsic function `Is_Negative` and, as a result, the procedure `Copy_Sign` has been totally reworked.

Because the new coding is backward compatible the problem described in the presentation has now effectively been fixed.

### 3.4 Static library

When all the sources have been modified, the static library can be built. To build the library we simply compiled all the `.adb` files that start with the prefix `Adam-` and then used the GCC archive program `Ar` to place the objects into a static library. This library is then cited as an input in the GCC linker script.

## 4 Returning unconstrained types

String handling in Modula-2 is very primitive. Strings are simply arrays of characters and because Modula-2 functions cannot return variable sized objects, they cannot therefore return strings.

Up until now we have had very little need for string manipulation. This is probably because our user interface is PC based rather than executing on any of the microprocessors. However, this would change if, for example, we implemented a stand-alone product that communicated with the user using TCP Telnet or as a web server using HTML. In such a project we would need improved string handling – such as that provided by Ada.

However, in order that Ada functions are able to return unconstrained types such as strings, we need to implement a secondary stack. This is a per-task area of memory from which the compiler can allocate space to return unconstrained types.

First the pragma `Restrictions (No_Secondary_Stack)` has to be removed from `System.ads` otherwise any function that attempts to return an unconstrained type will cause GNAT to issue the error message violation of implicit restriction "No\_Secondary\_Stack".

Once this restriction pragma has been removed from `System.ads`, GNAT will then issue the rather cryptic message

construct not allowed in configurable run-time mode

What this means is that GNAT has been unable to find the specification packages that enables returning unconstrained types. Just like with the enabling of floating-point these specifications need to be placed in the source search path and be contained within files with crunched names.

The required specification files are:

- `s-parame.ads`
- `s-secsta.ads`

- s-stoele.ads

Once these have been made available, code can be compiled.

However, linking will fail because of unresolved references.

A program that uses the secondary stack to return unconstrained types will need the routines:

- system\_\_secondary\_stack\_\_ss\_allocate
- system\_\_secondary\_stack\_\_ss\_mark
- system\_\_secondary\_stack\_\_ss\_release

We provide these routines in the same way as we did the floating-point routines, namely, in a static library. These routines are wrappers around our kernel functions to allocate memory from the calling task's secondary stack, to read its secondary stack pointer and to set its secondary stack pointer respectively.

In our Kernel, when a task is created, the size of its primary stack is given and optionally the size of a second stack. The primary stack is accessed via the processor's stack pointer register, and the secondary stack via access routines provided by the Kernel. Both stacks are protected by the Memory Protection Unit (MPU) and are private to the task. I.e. a memory fault will be raised should another task attempt to access them.

The Kernel provides three routines:

- AllocateFromSecondaryStack  
Takes the required size as a parameter and returns the start address of the memory area that has been allocated.
- GetSecondaryStackPointer  
Returns the secondary stack pointer of the current task.
- SetSecondaryStackPointer  
Sets the task's secondary stack pointer to the address passed as its parameter if the address is between the task's top of stack and the current secondary stack pointer.

These routines conveniently directly map onto the routines required by Ada.

## 5 String attributes

The attribute 'image, its short-hand form 'img and the attribute 'value are immensely useful when dealing with strings. To enable these, we need to provide GNAT with 12 specification packages all contained in files with crunched filenames.

We need 6 for the image attribute

- s-imgboo.ads
- s-imgllu.ads
- s-imgrea.ads
- s-imguns.ads
- s-imgint.ads
- s-imglli.ads

and a further six for the value attribute

- s-valboo.ads
- s-valint.ads
- s-valli.ads
- s-valllu.ads
- s-valrea.ads
- s-valuns.ads

At link time we need to resolve unsatisfied references, once again by use of a static library. Just as we did to support floating-point, we need to rename the Ada source files to the parent package Adam, compile them, and place the resultant objects into a static library.

This time there are 37 files:

- Adam-Image\_LLU.ads
- Adam-Image\_LLU.adb
- Adam-Image\_Uns.ads
- Adam-Image\_Uns.adb
- Adam-Img\_Bool.ads
- Adam-Img\_Bool.adb
- Adam-Img\_Int.ads
- Adam-Img\_Int.adb
- Adam-Img\_LLI.ads
- Adam-Img\_LLI.adb
- Adam-Img\_LLU.ads
- Adam-Img\_LLU.adb
- Adam-Img\_Real.ads
- Adam-Img\_Real.adb
- Adam-Img\_Uns.ads
- Adam-Img\_Uns.adb
- Adam-Float\_Control.ads
- Adam-Val\_Bool.ads
- Adam-Val\_Bool.adb
- Adam-Val\_Int.ads
- Adam-Val\_Int.adb
- Adam-Val\_LLI.ads
- Adam-Val\_LLI.adb
- Adam-Val\_LLU.ads
- Adam-Val\_LLU.adb
- Adam-Val\_Real.ads
- Adam-Val\_Real.adb
- Adam-Val\_Uns.ads
- Adam-Val\_Uns.adb
- Adam-Val\_Util.ads
- Adam-Val\_Util.adb
- Adam-Value\_LLU.ads
- Adam-Value\_LLU.adb
- Adam-Value\_Uns.ads



- Adam-Value\_Uns.adb
- Adam-Case\_Util.ads
- Adam-Case\_Util.adb

## 6 Images of Enumerations

However this isn't quite the whole story.

Left like this, obtaining the string representation of an enumeration literal does not bring back the uppercase version of the declaration but the string representation of its position within the enumeration.

For example:

```
type Color is (Blue, Green, Red);
C : Color := Red;
CI : constant String := C'image;
```

In the above example we would expect CI to be the 3 byte string "RED" whereas, in fact, it is the two byte string "2"!

I.e. GNAT has silently, and without any warning whatsoever, implemented

```
CI : constant String:= Color'pos(C) 'image;
```

rather than what was written!

For the correct implementation the name strings have to be retained. GNAT can be instructed to do this by commenting out or otherwise removing the pragma `Discard_Names` in `System.ads`.

The comment in `System.ads` for this pragma is

*"Disable explicitly the generation of names associated with entities in order to reduce the amount of storage used. These names are not used anyway."*

However, the last sentence in this comment is no longer true, the names, if they are there, can be used.

But just providing the names is not sufficient. Two more crunched specifications need to be provided:

- s-imenne.ads
- s-valenu.ads

along with their implementations:

- Adam-Img\_Enum\_New
- Adam-Val\_Enum

Then, and only then, does GNAT compile the correct code and the program work correctly and according to the Ada standard.

## 7 Protected Objects

As I mentioned in the description of the secondary stack, our system is multi-tasking, but we cannot use Ada's tasking model because ZFA precludes the use of Ada's run-time. However, our Kernel does support multi-tasking and therefore we need to protect certain data against simultaneous access.

In Modula-2, one way of doing this is by using a semaphore for each group of variables that need to be protected.

Unfortunately, this approach is somewhat error prone, it is all too easy to forget to write the code to wait on the group's semaphore, or even sometimes to use the wrong semaphore.

Ada makes this easier and less error prone with its protected object construct. Fortunately, although we don't use Ada's tasking model, ZFA supports the implementation of protected objects, albeit restricted to procedural operations. Protected entries are not supported because these would not work with our state machine model.

The following is an example of how we can use a protected type to ensure that the increment of a quadword is made task safe on a processor where 64-bit operations are not atomic.

```
type Value is mod 2**64;
```

```
protected Data is
  procedure Increment;
  function Actual_Value return Value;
private
  The_Value : Value := 0;
end Data;
```

```
protected body Data is
  procedure Increment is
  begin
    The_Value := The_Value + 1;
  end Increment;

  function Actual_Value return Value is
  begin
    return The_Value;
  end Actual_Value;
end Data;
```

If we try to compile this, GNAT will issue the by now familiar message

construct not allowed in configurable run-time mode

indicating that an Ada feature needs to be enabled by supplying a specific specification package within a file with a crunched filename.

To enable protected objects, we need to supply:

- s-taprob.ads
- s-taskin.ads

At link-time we will need to provide three routines

- `system__tasking__protected_objects__initialize__protection`
- `system__tasking__protected_objects__lock`
- `system__tasking__protected_objects__unlock`

Which we implement simply as wrapper routines around our Kernel's semaphore routines `Initialize`, `Wait_For` and `Signal` respectively.

## 8 Dynamic memory allocation

For example

```
C := new Character ('x');
```

In order that we can dynamically allocate memory using the Ada “new” construct we need to enable the feature by including the specification file `s-memory.ads`

At link time we must then satisfy the reference to `__Gnat_Malloc` with an implementation that simply calls our Kernel’s `Allocate` function that takes the desired amount of memory as a parameter and returns the address of the allocated memory.

## 9 Summary

In my first presentation [1] I described how Ada could be linked into an application predominantly written in C, how Ada routines could call C routines and C routines call Ada and how the Ada packages could be correctly elaborated.

This presentation goes further and shows that even more Ada features can be used if they are enabled by the simple

expedient of providing the required specification package in a file with the expected crunched filename. I also explained how the implementation could be provided by a static library and how this could be built. Together, these two techniques have allowed us to expand our use of Ada to include floating-point, functions returning strings, protected objects and memory allocation thereby making Zero Footprint Ada even more useful.

## References

- [1] A. Marriott and U. Maurer, “Using Ada in non-Ada Systems”, *Ada User Journal*, vol 39 no 3, pp 180-187, 2018.
- [2] Zip-Ada is a free, open-source programming library for dealing with the Zip compressed archive file format. <https://sourceforge.net/projects/unzip-ada> or <ssh://git@github.com/zertovitch/zip-ada>

# Static Analysis for Ada, C/C++ and Python: Different Languages, Different Needs

*Maurizio Martignano*

*Spazio IT – Soluzioni Informatiche s.a.s., San Giorgio Bigarello, Italy; email: Maurizio.Martignano@spazioit.com*

## Abstract

*Spazio IT has been working on the Independent Software Verification and Validation of several codebases, some written in Ada, others in C/C++ and more recently also in Python; in all cases Spazio IT has used static analysis techniques and tools facilitating code inspection. Static analysis has always proven to be beneficial, but depending on the programming language, its advantages have emerged in different areas and namely: metrics and structural analysis for Ada, bugs finding at execution/semantic level for C/C++ and errors finding at compilation/syntactical level for Python. The paper presents first some general considerations on static analysis, then it concentrates on static analysis specifically applied to Ada, C/C++ and Python codebases. Finally, the paper describes Spazio IT future activities in the area of static analysis.*

*Keywords: static analysis, code inspection, Ada, C/C++, Python, open source*

## 1 Introduction

Spazio IT has been working on several Independent Software Verification and Validation projects, especially in application domains like avionics and automotive, where the codebases under analysis have been written mostly in C/C++, sometimes in Ada and more recently also in Python. Though in all these projects static analysis has always proven to be beneficial, its advantages have emerged in different areas and namely: metrics and structural analysis for Ada, bugs finding at execution/semantic level for C/C++ and errors finding at compilation/syntactical level for Python.

Section two of this paper will present static analysis in general, why it is used in ISVV projects, in which areas it provides the more benefits and its relationships with testing. Section three will concentrate on static analysis and Ada. Section four on static analysis and C/C++. Section five will concentrate on static analysis and Python. Section six will present Spazio IT next activities in the area of static analysis.

## 2 Why Static Analysis?

Static analysis techniques and tools are used in Independent Software Verification and Validation mostly for three reasons:

- Metrics Gathering, Structure Analysis;
- Checking the adherence of the codebase under analysis to Guidelines and Standards;
- Bugs Finding.

### Metrics Gathering, Structure Analysis

In some projects the codebase under analysis has to satisfy a set of quality requirements, it has to “meet some specific set of quality objectives”. This is usually verified according to a particular software quality model, e.g. ISO/IEC 25010 [1]. These models identify some quality “characteristics” (e.g. “maintainability”, that derive from some sub-characteristics (e.g. in the case of “maintainability” its sub-characteristic are “modularity”, “reusability”, “analysability”, “modifiability”, “testability”). Sub-characteristics in turn derive/are computed from “metrics” like “lines of code”, “ratio comment to code”, “cyclomatic complexity”, “construct nesting”, and so on.

Metrics are gathered by static analyzers; some static analyzers are also able to compute sub-characteristics and, in turn, characteristics according to selected models.

In Spazio IT experience, several times, these “apparently theoretical” quality measurements, have allowed the identification of critical areas in the codebase actually requiring fixes to improve their readability and maintainability.

### Checking Guidelines and Standards

The adoption of guidelines and coding standards is supported by the principle that, given a programming language, it is possible to identify a subset of that language aiming at improving the portability, security and safety aspects of programs written in that language. MISRA C 2012, MISRA C++ 2008 [2] and AUTOSAR C++ 14 [3] are example of such “subsets” for C/C++. SPARK 2014 [4] is another example of these language “subsets”, but this time for Ada. While, at the moment, the C/C++ “subsets” can only be checked via static analyzers, SPARK 2014 stands on top of the GNAT compilation system.

In Spazio IT experience, only relating to C/C++ “subsets”, adopting these guidelines and standards has often generated a considerable amount of noise, false positives, especially when applied to legacy codebases. This is why in these cases, a fine-tuning activity, identifying which checks, rules to allow and which ones to disable before the actual analysis execution had been necessary.

### Bugs Finding

Static analysis has proven to be very helpful also in bugs finding. This capacity has been relying on techniques like abstract interpretation, data flow analysis during symbolic (i.e. virtual) execution, bounded model checking and the like.

With time most of these techniques, that originally were only available in some advanced commercial tools or in some kind or research/experimental tools, have been implemented also in “popular” and well supported tools like Clang Static Analyzer, Clang Tidy and FB-Infer. These “new generation” tools, being based on actual compiler technologies, are able to process real codebases in a reasonable time; on the contrary some of the original tools could only work on very limited portions of code.

### Syntactical Checks

A fourth reason, only appeared recently with “interpreted” languages like Python and justifying the use of static analysis, is that on “interpreted” codebases the static analyzer, the “linter” can be used as a sort of compiler to verify the syntactical correctness of the entire codebase, very difficult to prove otherwise. More on this in section five, on Python.

### Static Analysis vs Testing

Static analysis, especially bugs finding, has sometimes been considered as a replacement for testing. Spazio IT believes it is better to consider static analysis as a complementary activity to testing. The quality of testing depends on the quality and completeness of the test cases. Static analysis efficiency in bugs finding depends on the used techniques and tools and not on the quality of the test cases or their completeness. In huge codebases static analysis techniques could help identifying which areas require more testing than others, so they could help in maximizing the results that can be obtained with testing – also when resources are limited.

## 3 Static Analysis in Ada

Table 1 shows the Ada static analyzers used at Spazio IT over the years.

	Metrics Structure Analysis	Guidelines/ Standards	Bugs Finding (Abstract Interpretation, Symbolic Execution)
AdaControl	M	M	
GNAT tools	M GNATmetric	M GNATcheck	A CodePeer
PolySpace for Ada	M	M	A
Understand	M	M	

**Table 1. Ada Static Analyzers used at Spazio IT (M = mostly used for, A = also used for)**

AdaLog AdaControl [5] has been used for metrics gathering and checking the adherence of the codebases to project-specific coding standards. AdaCore GNATmetric [6] and GNATcheck [7] have also been used for metrics gathering and checking the adherence of the codebases to standards like DO-178B/C [8] and EN 50128 / IEC 62279 [9]. Likewise, for the very same two reasons, also MathWorks PolySpace for Ada [10] and SciTools Understand [11] have been used.

In some limited cases AdaCore CodePeer [12] and Polyspace have also been used as supporting tool when looking for bugs during coding inspections activities.

In 2015 Spazio IT developed for Airbus an Ada SonarQube Plugin able to compute the so called “Maintainability Index” based on a quality model developed by Airbus and similar to ISO/IEC 25010; in this model “maintainability” was a function of “analyzability”, “changeability”, “stability”, and “testability”.

## 4 Static Analysis in C/C++

Table 2 shows the C/C++ static analyzers used at Spazio IT over the years.

	Metrics Structure Analysis	Guidelines/ Standards	Bugs Finding (Abstract Interpretation, Symbolic Execution)
PC-Lint-Plus	M	M	A
Cppcheck	A	A	M
Clang-SA Clang-Tidy			M
FB-Infer			M

**Table 2. C/C++ Static Analyzers used at Spazio IT (M = mostly used for, A = also used for)**

Gimpel Software PC-Lint(-Plus) [13] has been used mostly for metrics gathering and checking the adherence of the codebases to MISRA and AUTOSAR coding standards. In some cases, PC-Lint has also been used for bugs finding (e.g. identification of recursive loops, logical errors in data flow processing and the like).

The open-source tools Cppcheck [14], Clang-SA (Static analyzer) [15], Clang-Tidy [16] and FB Infer [17] have been used for bugs finding. Cppcheck has also been used for metrics gathering and checking the adherence of the codebases to MISRA.

It has to be noticed that, recognizing the power and continuous improvement of the Clang tools, also Cppcheck is more and more relying not only on its internal analysis capabilities but on these Clang tools.

Figure 1 shows that C/C++ compilers, if properly used, can be as strict as Ada compilers (e.g. Ada Gem #33 [18]).

In the Clang compilation system, the compiler and the static analyzers (Clang-SA and Clang-Tidy) are based on the very same libraries, so, in a way, the “Compiler” is the “Static Analyzer”.

```

C:\SMART\PROG\ProveAda>gnat compile Static_Check.adb
gcc -c static_check.adb
static_check.adb:8:17: non-local pointer cannot point to local object
gnatmake: "static_check.adb" compilation error

C:\SMART\PROG\ProveAda>clang -o dp.exe -Wall -Werror dp.c
dp.c:18:11: error: address of stack memory associated with local variable 'x'
returned [-Werror, -Wreturn-stack-address]
    return &x;
           ^
1 error generated.

C:\SMART\PROG\ProveAda>

```

**Figure 1. Dangling references spotted by the Ada compiler and by the C compiler**

Static analyzers as CBMC [19] and Frama-C [20] though very interesting from a theoretical and research point of view, at the moment, that is April 2021, are not a viable option for the analysis of large codebases. It is also interesting to notice that, like Cppcheck, also Frama-C is more and more trying to integrate with the Clang compilation system and static analyzers.

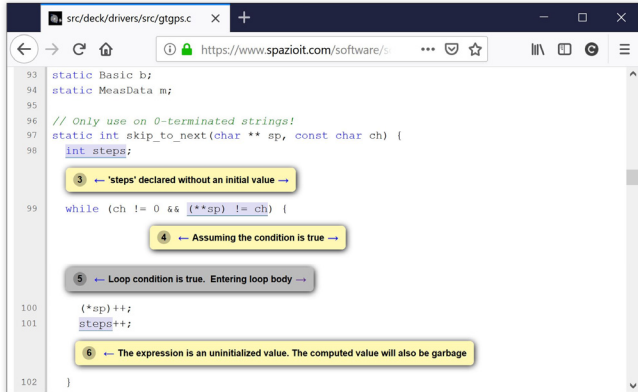


Figure 2. A bug discovered by Clang Static Analyzer and showed in its GUI

Figure 2 shows a bug caught by Clang Static Analyzer and Figure 3 shows the very same bug, this time displayed from by SonarQube (a code quality platform used by Spazio IT for all its ISVV activities [21]).

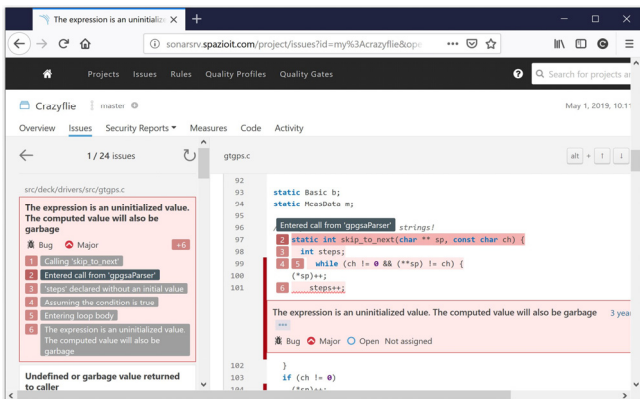


Figure 3. The very same bug, caught by Clang Static Analyzer and shown by SonarQube

### 5 Static Analysis in Python

Table 3 shows the Python static analyzers used at Spazio IT in the recent years.

	Metrics Structure Analysis	Guidelines / Standards	Bugs Finding (Logical Errors, Security Issues, Abstract Interpretation)	Compilation
Bandit	A	A	M	
Flake8	M	M		
Pylint	A	A	M	A
SQ Python Plugin	A	A	M	

Table 3. Python Static Analyzers used at Spazio IT (M = mostly used for, A = also used for)

The open-source tool Flake8 [22] has been used for metrics gathering and checking the adherence of the codebases to some project-specific coding standards.

Like in the case of C/C++, also in Python, the open-source tools Bandit [23], Pylint [24] and SonarQube Python Plugin [25] have been used for bugs finding.

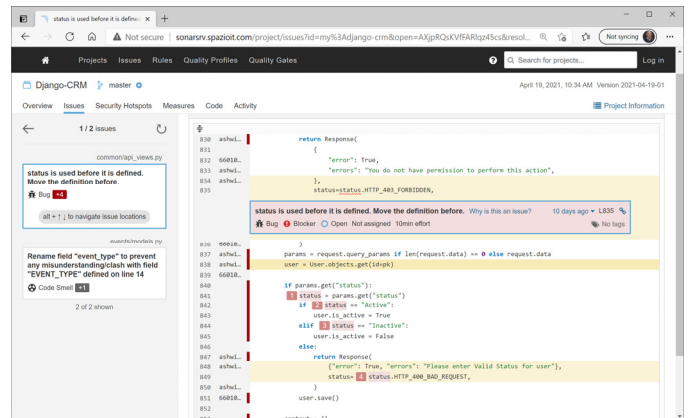


Figure 4. SonarQube platform showing a bug caught by the SonarQube Python Plugin

Somehow static analysis is applied to Python codebases in the same way that is applied to C/C++ codebases: mostly for bugs finding. But Python as also an additional “use case” for static analysis.

### The Linter as Compiler



Figure 5. An incorrect piece of code in Python

Figure 5 shows a piece code written in Python, where there’s an obvious error. This error (“pront” instead of “print”), if Python is invoked in “interpreted mode” would manifest itself only if line 6 is executed. On the contrary a similar situation would not occur in a compiled language where the error would not escape the compilation / linking phase.

Many Python codebases, e.g. all web application based on the “Django” framework [26] start with statements similar to the following:

```
>&2 echo "Migrations complete."

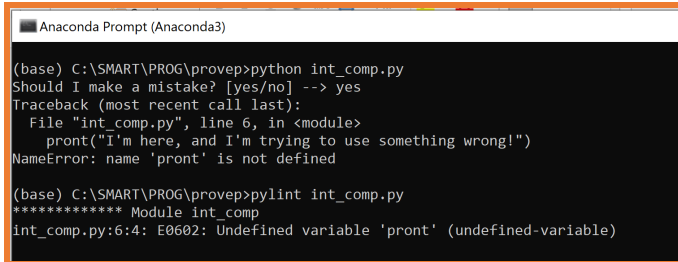
python -u manage.py runserver 0.0.0.0:8080
```

Figure 6. The starting point of a Django base application

So, the application is started with Python in interpreted mode.

If the application has a huge codebase, it might very well be that inside these codebases there could be some syntactical (or link type) errors that only the actual execution or a sort of “compilation” can catch.

Figure 7 shows that Pylint, a static analyser, can also perform these “compilation/link type” checks before actual execution.



```

Anaconda Prompt (Anaconda3)

(base) C:\SMART\PROG\provep>python int_comp.py
Should I make a mistake? [yes/no] --> yes
Traceback (most recent call last):
  File "int_comp.py", line 6, in <module>
    pront("I'm here, and I'm trying to use something wrong!")
NameError: name 'pront' is not defined

(base) C:\SMART\PROG\provep>pylint int_comp.py
***** Module int_comp
int_comp.py:6:4: E0602: Undefined variable 'pront' (undefined-variable)

```

Figure 7. The linter as “compiler”

## 6 Spazio IT future activities on Static Analysis

Spazio IT, apart from continuing being involved in ISVV projects, in the next few months/years will be busy:

1. Keeping up to date with the evolution of Clang Static Analyzer, Clang-Tidy and FB Infer.
2. Keeping up to date with the evolution of SonarQube.
3. Improving Spazio IT SAFe Toolset [27], by adding to it support for Python.

The SAFe (Static Analysis Framework) Toolset is a set of open-source tools, packaged in easily reusable form (currently, an Ubuntu Virtual Machine), that can be used to perform Software Verification and Validation.

### References

- [1] <https://www.iso.org/standard/35733.html>
- [2] <https://misra.org.uk/Publications/tabid/57/Default.aspx>

- [3] [https://www.autosar.org/fileadmin/user\\_upload/standards/adaptive/20-11/AUTOSAR\\_RS\\_CPP14Guidelines.pdf](https://www.autosar.org/fileadmin/user_upload/standards/adaptive/20-11/AUTOSAR_RS_CPP14Guidelines.pdf)
- [4] <https://github.com/AdaCore/spark2014>
- [5] <https://www.adalog.fr/en/adacontrol.html>
- [6] [https://learn.adacore.com/courses/GNAT\\_Toolchain\\_Intro/chapters/gnat\\_tools.html#gnatmetric](https://learn.adacore.com/courses/GNAT_Toolchain_Intro/chapters/gnat_tools.html#gnatmetric)
- [7] <https://www.adacore.com/gnatpro/toolsuite/gnatcheck>
- [8] <https://www.rtca.org/>
- [9] <https://webstore.iec.ch/publication/22781>
- [10] <https://www.mathworks.com/products/polyspace-ada.html>
- [11] <https://www.scitools.com/>
- [12] <https://www.adacore.com/codepeer>
- [13] <https://gimpel.com/>
- [14] <http://cppcheck.sourceforge.net/>
- [15] <https://clang-analyzer.llvm.org/>
- [16] <https://clang.llvm.org/extra/clang-tidy/>
- [17] <https://fbinfer.com/>
- [18] <https://www.adacore.com/gems/gem-33>
- [19] <http://www.cprover.org/cbmc/>
- [20] <https://frama-c.com/>
- [21] <https://www.sonarqube.org/>
- [22] <https://flake8.pycqa.org/>
- [23] <https://pypi.org/project/bandit/>
- [24] <https://pylint.org/>
- [25] <https://github.com/SonarSource/sonar-python>
- [26] <https://www.djangoproject.com/>
- [27] [https://spazioit.com/pages\\_en/sol\\_inf\\_en/code\\_quality\\_en/safe-toolset-en/](https://spazioit.com/pages_en/sol_inf_en/code_quality_en/safe-toolset-en/)



# ASIS vs. LibAdalang: A Comparative Assessment

*J-P. Rosen*

*Adalog, 2 rue du Docteur Lombard, 92130 Issy-Les-Moulineaux, France; email: rosen@adalog.fr*

## Abstract

*This paper compares the origins, features, and status of two different tools intended to facilitate static analysis of Ada programs: ASIS and LibAdalang.*

*It stresses the differences in principles, features, and intended usages, and shows use cases where each is more appropriate.*

*Keywords: Ada, static analysis, ASIS, LibAdalang.*

## 1 Introduction

Ada is a language which is both large and complex to compile. Any tool that aims at providing some form of static analysis of an Ada program must cope with visibility rules, overloading resolution, generic instantiations, defaulted parameters, etc.

For example, consider the simple statement:

```
V := A (B);
```

Possible interpretations are:

- A is a function, B is the parameter
- A is an array, B is an index
- A is a type, B is converted to A
- A is a parameterless function call returning an array, B is an index
- A is a pointer to a function, B is the parameter

And of course, B can be a constant, a variable, or a function call...

Starting an analysis tool from scratch would require almost rewriting half of a compiler. To avoid this effort and foster the development of language tools, ASIS was developed as an API to access the decorated AST (*Abstract Syntax Tree*) produced by the compiler. More recently, AdaCore [2] developed an alternative solution named LibAdalang.

Developers of Ada tools have now two competing solutions, and are faced with fundamental decisions:

- When developing a new tool, which solution is more appropriate?
- When evolving an ASIS tool, is it appropriate to invest time and money for moving it to LibAdalang?

This paper aims at providing some material to help answering these questions.

## 2 Origin

### ASIS

ASIS [1] is an international standard, first developed for Ada 83, then updated to Ada 95. It was developed by an international committee (the ASIS working group), and built upon experience gained from previous attempts to standardize an intermediate representation for Ada, notably DIANA [3].

Several compilers provide the ASIS interface. The standard was not evolved for subsequent updates of Ada, however implementations, especially the AdaCore one, continued to add support up to Ada 2012. AdaCore announced however that their implementation would not be upgraded to support Ada 2022.

### LibAdalang

LibAdalang is an independent, self-funded development of AdaCore. It is developed by a dedicated team from AdaCore. It is an open-source project<sup>1</sup> available on GitHub [2], and the team welcomes comments from the community; however, there is no control of any official or international body over the design decisions.

## 3 Fundamental principles

### ASIS

ASIS is an API to explore and get information from the decorated syntactic tree, as produced by the associated compiler. This guarantees that a tool based on ASIS will see the code exactly as the compiler sees it, including implementation dependent elements allowed by the standard, and elements defined in the System and Standard packages. However, this implies that an ASIS implementation is linked to a certain compiler. A tool based on ASIS must provide specific versions for each supported compiler.

Since ASIS operates on a tree resulting from a successful compilation, it cannot handle incorrect or incomplete code. For the same reason, it was a deliberate design decision to *not* provide any operation that would modify, or even add information, to the syntactic tree. It is purely oriented towards analysing a program, with no way to modify it.

### LibAdalang

LibAdalang includes its own analyser and tree constructor, which is part of the library, and embedded with any application that uses it. A LibAdalang application is therefore stand-alone, independent of any compiler, and can

<sup>1</sup> This is assumed from the long-time involvement of AdaCore with free software. However, at the time of writing, the project distribution has no mention saying “LibAdalang is free software”; this is likely to be an omission, but it makes the copyright status unclear

be used even without an Ada compiler on the target machine. On the other hand, there is no guarantee that the view of the program provided to the tool is strictly equivalent to the compiler's view, nor that packages `System` and `Standard` match the ones of the compiled code. More precisely, LibAdalang will happily accept any program which is *syntactically* correct, even if it is not *semantically* correct. For example, no error is diagnosed in the following program:

```

procedure Incorrect is
  I : Integer;
begin
  I := 1.0; -- Typing error
end Incorrect;

```

A goal of LibAdalang was to be usable in contexts such as syntactic editors, where the source can be incorrect, and to be able to fix such incorrect code or to automatically complete it. Of course, there are limitations to what can be achieved on incorrect code, since almost nothing can be assumed. LibAdalang provides operations to modify the underlying tree and the original source.

Moreover, LibAdalang provides a Python interface for developing rapid applications or interactively trying the interface.

## 4 Style and organization

### ASIS

ASIS has a root package (`Asis`) that defines the basic entities used by the rest of the API. Child units group queries according to the structure (chapter and clauses) of the Ada reference manual: `Asis.Declarations`, `Asis.Definitions`, `Asis.Expressions`, `Asis.Statements`, etc.

Other packages are provided for initialization and loading of compilation units, accessing the source text of any element, etc.

The tree managed by ASIS is strictly the abstract syntax tree (AST) as defined by the syntax of the language; concrete elements that are not syntactic elements (comments, semi-colons, spaces) do not appear in the tree. It is possible to access the source corresponding to a node in the tree, but only as text. This makes it more complicated to develop applications like pretty printers that deal mainly with the physical appearance of the program [4].

### LibAdalang

LibAdalang provides only two packages related to analysis: `Libadalang.Common` and `Libadalang.Analysis`. The root package `Libadalang` is almost empty and serves only as the parent of the hierarchy. `Libadalang.Common` groups general declarations of types and subprograms used in the rest of the API, including declarations intended only for the implementation of the library, and not for the user of LibAdalang. `Libadalang.Analysis` gathers all syntactic and semantic queries in a single package; as of this paper, the specification of `Libadalang.Analysis` contains 15992 raw lines, including 14154 lines in the visible part.

Other packages are provided for initialization and loading of compilation units, changing program text on the fly, etc.

In addition to syntactic elements, LibAdalang keeps all syntactic tokens, including *trivias* representing the concrete representation, like spacing in the source, semi-colons, comments, etc. The goal is to be able to manipulate the concrete representation of the program as well as its abstract structure.

## 5 Typing system

### ASIS

All Ada elements are of a single type: `Element`. Subtypes are provided for documentation purposes, like:

```

subtype Declaration is Element;

```

but since there are no constraints, there is no compile-time check that an element belongs to the expected subtype. The hierarchy of syntactic elements is accessible through a number of classification functions returning enumeration types that tell the precise “kind” of the element. For example, the function `Element_Kind` returns a value like `A_Declaration`, `An_Expression`, `A_Statement`... If the element is a declaration, the function `Declaration_Kind` returns `A_Subtype_Declaration`, `A_Variable_Declaration`, `A_Constant_Declaration` ...

Every query expects its argument to be of certain kinds, and checks it at run-time (and raises the exception `Inappropriate_Element` if the checks fails). This means that ASIS is strongly, but dynamically typed.

This comes as a surprise to many Ada users who are more accustomed to static strong typing. However, this simplifies navigating through the syntactic tree, since it is often not possible to foresee the exact nature of the result of a query. For example, it is very common in tools to navigate “up” the tree. A simple loop to find which procedure body encloses a certain element can be done simply, thanks to dynamic typing:

```

declare
  E : Asis.Element := Some_Query (...);
begin
  while declaration_kind (E)
    not in A_Procedure_Body_Declaration
  loop
    E := Enclosing_Element (E);
  end loop;
end;

```

### LibAdalang

LibAdalang represents elements as a hierarchy of tagged types, rooted at `Ada_Node`:

```

type Ada_Node is tagged private;
type Expr is new Ada_Node with private;
type Name is new Expr with private;
...

```

The whole hierarchy of elements contains 373 different types. Queries require parameters of the appropriate type, but return values of the *specific* type `Ada_Node`. These values must be converted to the appropriate type using ad-hoc conversion functions, which raise an exception if their

argument is not of the appropriate kind. Here is a typical example of this programming pattern:

```

case Kind (Node) is
  when Ada_Call_Expr =>
    for Assoc of As_Assoc_List
      (F_Suffix (As_Call_Expr (Node)))
    loop
      ...
    end loop;
  ...
end case;

```

Here, Node is obtained from a previous query, and is of type Ada\_Node, and its real kind is obtained by the Kind function. Since F\_Suffix expects a parameter of type Call\_Expr'Class, it must be converted by the special function As\_Call\_Expr. Of course, this function will raise an exception (Constraint\_Error) if the parameter does not correspond to the expected type.

Despite the apparent stronger typed hierarchy of elements, LibAdalang is also dynamically typed: if a node does not belong to the expected kind for a query, it will be checked at run-time by the corresponding "As\_..." function instead of the function itself. On the other hand, the stronger typing makes exploring the tree more difficult; for example, the simple loop of the previous example has to be replaced by the following recursive function:

```

function Enclosing_Proc (N : Ada_Node'Class)
  return Ada_Node'Class
is
begin
  if Kind (N) in
    Ada_Subp_Kind_Procedure | Ada_Subp_Body
  then
    return N;
  else
    return Enclosing_Proc (Parent (N));
  end if;
end Enclosing_Proc;

```

## 6 Tree traversal

Tree traversal is the process by which a program explores the whole program under analysis.

### ASIS

ASIS provides a generic traversal function, which must be instantiated to provide the actual traversal function:

```

generic
  type State_Information is limited private;
  with procedure Pre_Operation
    (Element : Asis.Element;
     Control : in out Traverse_Control;
     State : in out State_Information)
  is <>;
  with procedure Post_Operation
    (Element : Asis.Element;
     Control : in out Traverse_Control;
     State : in out State_Information)

```

```

is <>;
procedure Traverse_Element
  (Element : Asis.Element;
   Control : in out Traverse_Control;
   State : in out State_Information);

```

The Pre\_Operation procedure is called when entering a node in the AST, before traversing the children, while the Post\_Operation procedure is called when returning to the node after traversing the children. In addition, a variable of the (user provided) type State\_Information is passed along. This makes it convenient to initialize information when reaching a node, gathering information while traversing the children, and using the result when returning to the node.

Each of the operations returns a value of the enumeration type State\_Information; possible values are Continue (normal case), Abandon\_Children (child nodes are not traversed), Abandon\_Siblings (return immediately to the parent node), and Terminate\_Immediately.

### LibAdalang

LibAdalang provides a traversal function where the processing of the node is provided as a pointer to an appropriate function:

```

function Traverse
  (Node : Ada_Node'Class;
   Visit : access function (Node : Ada_Node'Class)
     return Visit_Status)
  return Visit_Status;

```

The Visit function is called when entering a node in the AST, before traversing the children. The enumeration type Visit\_Status can take the values Into (continue normally), Over (child nodes are not traversed) and Stop. There is no equivalent to Abandon\_Siblings.

There is no provided function to perform processing when returning to the node after traversing the children; if this is desired, the Visit function must manually invoke Traverse on child nodes and add the necessary processing after it returns. Since there is no associated data, information must be gathered in a global variable, or equivalent data structure.

## 7 Documentation

### ASIS

The official documentation is the ASIS standard itself. It provides a good description of how to build an ASIS application and examples, in addition to the API itself. But being an ISO standard, it is a copyrighted document that must be bought from ISO at usual ISO price (currently, CHF 198).

However, AdaCore's implementation comes with an ASIS user guide that provides appropriate guidance on how to build an ASIS application.

The API itself is self-documented. The naming convention of the structural queries follows strictly the names and the syntax used in the ARM, making it easy to find the desired function. For example, the syntax of an assignment in the ARM is given as:

```
variable_name := expression;
```

The corresponding structural queries will be:

```
function Assignment_Variable_Name
  (Statement : Asis.Statement)
  return Asis.Expression;
function Assignment_Expression
  (Statement : Asis.Statement)
  return Asis.Expression;
```

Each query states precisely the kinds of expected elements, and the kinds of the provided result. For example, the comments on the above `Assignment_Variable_Name` function state:

```
-- Statement - Specifies the assignment statement to query
-- Returns the expression that names the left hand side of the
-- assignment.
-- Appropriate Element_Kinds:
--   A_Statement
-- Appropriate Statement_Kinds:
--   An_Assignment_Statement
-- Returns Element_Kinds:
--   An_Expression
```

Structural queries have names starting with “Corresponding\_”, making it easy to read and understand. For example, the query used to find the declaration of a given name is called `Corresponding_Name_Declaration`.

### LibAdalang

LibAdalang comes with a user guide, providing a detailed explanation of how to create an application, both in Python and Ada.

The API has a layout that shows that it is still work in progress: no header comments, lots of useless blank lines, poor indentation... The naming convention of queries bears no relationship to the reference manual; for example, the syntactic element `Selector` in Ada is called `Suffix` in LibAdalang. Many names use abbreviations whose meaning is far from obvious, when not misleading. For example, the query that returns the list of names that follow a `with` clause is named `F_Packages`... although a `with` clause can refer to units that are not packages!

Queries are divided into those that return “fields” of the underlying structure (i.e. structural queries) and those that return “properties” (i.e. semantic queries). The first ones have names that start in `F_` and the second ones have names that start in `P_`, a convention that may be useful to the implementation, but makes the reading quite unnatural. The documentation (in the accompanying document or in the comments in the package) is often missing, or simply states what kind of nodes is contained in the corresponding field - often with misleading information. For example, the documentation of the `F_Dest` query for an assignment (the left-hand side of the assignment) mentions as possible fields `Attribute_Ref`, `Char_Literal`, and `String_Literal`, while actually these cannot appear as the destination of an assignment statement!

## 8 Extra functionalities

### ASIS

The ASIS standard defines just an API. Helper utilities can be provided by the implementation, but there is no requirement to do so.

The AdaCore implementation provides a utility called `Asistant` that allows exercising interactively all queries of the API. It is very useful to understand the exact behaviour of the queries, but it is not intended as a way of quickly analysing a program.

As part of the distribution of `AdaControl` [5], `Adalog` provides a small utility called `ptree` that prints a semi-graphical representation of the syntactic tree, as obtained from ASIS; this is handy in understanding the structure of the AST.

### LibAdalang

LibAdalang provides a Python API in addition to the Ada API. This can be used to exercise queries as well as for developing quickly and interactively small analysis tasks. Of course, the Ada programmer will prefer the Ada interface for applications with a long lifetime...

For quick development of a command line application, LibAdalang provides the generic package `App`, which can be instantiated with a procedure traversing the tree, and sets up automatically all the environment, including command line parameters analysis, setting of the environment, etc.

An associated project is `LKQL` (*LangKit Query Language*), a language intended to provide language queries on top of LibAdalang.

## 9 Pros and Cons

### ASIS

*Pros:* ASIS works on legal code, after analysis by an Ada compiler that passes the validation. This provides great confidence that the program is analysed in conformance with the standard, and that the content of packages `Standard` and `System` match the ones used by the compiler. It comes with a complete documentation in the API, strongly linked to the Ada Reference Manual, making it easy to retrieve the necessary queries and to understand its effects.

*Cons:* The fact that ASIS works only on legal code and cannot change the tree or the corresponding source makes it inappropriate for interactive applications, such as IDEs, where the source is incomplete or evolving. Although AdaCores’s implementation processes all versions of Ada up to Ada 2012, an update of the standard to the latest version of Ada would be welcome.

### LibAdalang

*Pros:* LibAdalang is able to work on incomplete/incorrect code and provides sophisticated support to the concrete representation of the program, as well as editing and modifying the original text. It processes the latest version of the language.

*Cons:* As there is no connection to the compiler, there is no guarantee that LibAdalang's view of the program corresponds the compiler's view or to the Ada standard. An analysis tool cannot rely on the fact that there is no diagnosis to trust that Ada rules are being obeyed; therefore the tool should be run only on programs that have been successfully compiled with a full Ada compiler.

The typing system and the distance between Ada's formal definition and the analysis packages, as well as the lack of a number of useful features, makes it less fit for deep analysis of Ada code.

The development of LibAdalang is fully under control of AdaCore without external review, and of course it is not a standard, nor expected to become one.

## Conclusion

ASIS and LibAdalang cover different parts of the spectrum of code analysis tools. ASIS, thanks to its precise specification, its close connection to the Ada definition, and its guarantee of semantic correctness of the analysed code, is more appropriate for long lived tools, and especially tools expected to be used in demanding domains like instrumentation or control of safety critical software, The dynamic aspect of LibAdalang is well suited for all the tasks that require close connection to the source, user interaction, or dynamic modifications, like syntactic editors, code generators, and code transformation tools.

Given the fundamental differences in philosophy and typing systems between ASIS and LibAdalang, moving a tool from ASIS to LibAdalang cannot be done easily and would require a complete rewrite.

## References

- [1] ISO/IEC 15291: Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)
- [2] <https://github.com/AdaCore/libadalang>
- [3] G. Goos and G. Winterstein, "Towards a compiler front-end for Ada", *Proceedings of the ACM-SIGPLAN symposium on Ada programming language*, Annual International Conference on Ada. ACM-SIGPLAN. pp. 36–46, 1980.
- [4] S. Rybin and A. Strohmeier, " About the Difficulties of Building a Pretty-Printer for Ada", *Reliable Software Technologies — Ada-Europe 2002 Conference*, June 2002.
- [5] <https://github.com/Adalog-fr/Adacontrol>
- [6] [https://github.com/AdaCore/langkit-query-language/blob/master/user\\_manual/source/language\\_reference.rst](https://github.com/AdaCore/langkit-query-language/blob/master/user_manual/source/language_reference.rst)

# National Ada Organizations

## Ada-Belgium

attn. Dirk Craeynest  
c/o KU Leuven  
Dept. of Computer Science  
Celestijnenlaan 200-A  
B-3001 Leuven (Heverlee)  
Belgium  
Email: [Dirk.Craeynest@cs.kuleuven.be](mailto:Dirk.Craeynest@cs.kuleuven.be)  
URL: [www.cs.kuleuven.be/~dirk/ada-belgium](http://www.cs.kuleuven.be/~dirk/ada-belgium)

## Ada in Denmark

attn. Jørgen Bundgaard  
Email: [Info@Ada-DK.org](mailto:Info@Ada-DK.org)  
URL: [Ada-DK.org](http://Ada-DK.org)

## Ada-Deutschland

Dr. Hubert B. Keller  
Karlsruher Institut für Technologie (KIT)  
Institut für Angewandte Informatik (IAI)  
Campus Nord, Gebäude 445, Raum 243  
Postfach 3640  
76021 Karlsruhe  
Germany  
Email: [Hubert.Keller@kit.edu](mailto:Hubert.Keller@kit.edu)  
URL: [ada-deutschland.de](http://ada-deutschland.de)

## Ada-France

attn: J-P Rosen  
115, avenue du Maine  
75014 Paris  
France  
URL: [www.ada-france.org](http://www.ada-france.org)

## Ada-Spain

attn. Sergio Sáez  
DISCA-ETSINF-Edificio 1G  
Universitat Politècnica de València  
Camino de Vera s/n  
E46022 Valencia  
Spain  
Phone: +34-963-877-007, Ext. 75741  
Email: [ssaez@disca.upv.es](mailto:ssaez@disca.upv.es)  
URL: [www.adaspain.org](http://www.adaspain.org)

## Ada-Switzerland

c/o Ahlan Marriott  
Altweg 5  
8450 Andelfingen  
Switzerland  
Phone: +41 52 624 2939  
e-mail: [president@ada-switzerland.ch](mailto:president@ada-switzerland.ch)  
URL: [www.ada-switzerland.ch](http://www.ada-switzerland.ch)



# Ada-Europe Sponsors

---

**Ada Edge**

27 Rue Rasson  
B-1030 Brussels, Belgium  
Contact: Ludovic Brenta  
ludovic@ludovic-brenta.org

**AdaCore**

46 Rue d'Amsterdam  
F-75009 Paris, France  
Contact: Jamie Ayre  
sales@adacore.com  
www.adacore.com



2 Rue Docteur Lombard  
92441 Issy-les-Moulineaux Cedex  
France  
Contact: Jean-Pierre Rosen  
rosen@adalog.fr  
www.adalog.fr/en/

**Capgemini engineering**

22 St. Lawrence Street  
Southgate, Bath BA1 1AN  
United Kingdom  
www.capgemini.com



4545 E. Shea Blvd. #210  
Phoenix, AZ 85028  
USA  
Contact: Laurent Meilleur  
sales@ddci.com  
www.ddci.com



Jacob Bontiusplaats 9  
1018 LL Amsterdam  
The Netherlands  
Contact: Wido te Brake  
wido.tebrake@deepbluecap.com  
www.deepbluecap.com



24 Quai de la Douane  
29200 Brest, Brittany  
France  
Contact: Pierre Dissaux  
pierre.dissaux@ellidiss.com  
www.ellidiss.com



In der Reiss 5  
D-79232 March-Buchheim  
Germany  
Contact: Frank Piron  
info@konad.de  
www.konad.de

**PTC®  
Developer Tools**

3271 Valley Centre Drive,  
Suite 300  
San Diego, CA 92069, USA  
Contact: Shawn Fanning  
sfanning@ptc.com  
www.ptc.com/developer-tools



Signal Business Centre  
2 Innotec Drive, Bangor  
North Down BT19 7PD  
Northern Ireland, UK  
enquiries@sysada.co.uk  
www.sysada.co.uk



1090 Rue René Descartes  
13100 Aix en Provence, France  
Contact: Patricia Langle  
patricia.langle@systemel.fr  
www.systemel.fr/en/



Tiirasaarentie 32  
FI 00200 Helsinki, Finland  
Contact: Niklas Holsti  
niklas.holsti@tidorum.fi  
www.tidorum.fi

**VECTOR** 

Corso Sempione 68  
20154 Milano  
Italy  
Contact: Massimo Bombino  
massimo.bombino@vector.com  
www.vector.com



Beckengässchen 1  
8200 Schaffhausen  
Switzerland  
Contact: Ahlan Marriott  
admin@white-elephant.ch  
www.white-elephant.ch

**XGC Technology**

United Kingdom  
Contact: Chris Nettleton  
nettelton@xgc.com  
www.xgc.com

