ADA USER JOURNAL

Volume 40
Number 1
March 2019

Contents

Quarterly News Digest

Kristoffer Nyborg Gregertsen
SINTEF, Email: kristoffer.gregertsen@sintef.no

Contents

Ada-related Tools

Qt5Ada

From: leonid.dulman@gmail.com
Subject: Announce: Qt5Ada version 5.12.0 release 21/12/2018 free edition
Newsgroups: comp.lang.ada
Date: Fri, 21 Dec 2018 10:56:14 -0800

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.12.0 final)

Qt5ada version 5.12.0 open source and qt5c.dll, libqt5c.so(x64) built with Microsoft Visual Studio 2017 in Windows, gcc x86-64 in Linux.

Package tested with gnat gpl 2012 ada compiler in Windows 32bit and 64bit , Linux x86-64 Debian 9.4.

It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors, Bluetooth, Navigation and many others thinks.

Changes for new Qt5Ada release:

Added new packages: Qt.QStringView, Qt.QGraphicsCustomItem, Qt.QGLContext

My configuration script to build Qt 5.12.0 is: configure –opensource -release -nomake tests -opengl dynamic -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked OPENSSL_LIBS="-lssleay32 -llibeay32" -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -icu -prefix "e:/Qt/5.12"

As a role Ada is used in embedded systems, but with QTADA(+VTKADA) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing , Modbus control and many others thinks.

Qt5Ada and VTKAda for Windows, Linux (Unix) https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/

The full list of released classes is in "Qt5 classes to Qt5Ada packages relation table.docx" VTKAda version 8.1.0 is based on VTK 8.1.0 (OpenGL2) is fully compatible with Qt5Ada 5.12.0

I hope Qt5Ada and VTKAda will be useful for students, engineers, scientists and enthusiasts

With Qt5Ada you can build any applications and solve any problems easy and quickly.

If you have any problems or questions, tell me know.

AWS issue

From: Andrew Shvets <andrew.shvets@gmail.com>
Subject: Can't get to include AWS
Newsgroups: comp.lang.ada
Date: Thu, 27 Dec 2018 19:58:14 -0800

I installed the latest GNAT Community distribution from AdaCore in ~/GNAT and when I tried to use my *.GPR file in order to build my code, I encountered the below error:

unknown project file: "aws"

In my *.GPR file I did 'with "aws";'.

Is there some path or some other config value that needs to be set?

Thanks in advance for your replies.

From: eduardsapotski@gmail.com
Subject: Re: Can't get to include AWS
Newsgroups: comp.lang.ada
Date: Fri, 28 Dec 2018 01:23:55 -0800

Run GPS.

Open project.

Edit -> Project Properties -> Dependencies

Drag AWS to left panel.

Save.

Or in .gpr file paste: with "aws.gpr";

From: Simon Wright <simon@pushface.org>
Subject: Re: Can't get to include AWS
Newsgroups: comp.lang.ada
Date: Sat, 29 Dec 2018 20:30:30 +0000

> I installed the latest GNAT Community distribution from AdaCore in ~/GNAT and when I tried to use my *.GPR file in order to build my code, I encountered the below error:

> unknown project file: "aws"

> In my *.GPR file I did 'with "aws";'.

I have GNAT CE installed under /opt/gnat-ce-2018.

If I don't have /opt/gnat-ce-2018/bin on my PATH but say /opt/gnat-ce-2018/bin/gprbuild -P shvets.gpr where shvets.gpr contains 'with "aws";' I get the same as you.

If I do have /opt/gnat-ce-2018/bin on my PATH and say

```
gprbuild -P shvets.gpr
```

it works fine.

Protobuff for Ada

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Subject: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Fri, 28 Dec 2018 19:57:40 +0100

I managed to resurrect an old master thesis work that was done by Niklas Ekendahl in 2013 and put it on

https://github.com/persan/protobuf-ada

the plan is to get it in working shape.

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Fri, 28 Dec 2018 21:53:55 -0800

Cool!

More libs, bindings, and implementations in Ada is a good thing.

Though, it should be noted that ASN.1 is *probably* the better technology in cases where ProtoBufs are being considered:

http://ttsiodras.github.io/asn1.html

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Sat, 29 Dec 2018 12:05:40 +0100

> Though, it should be noted that ASN.1 is *probably* the better technology in cases where ProtoBufs are being considered:

> http://ttsiodras.github.io/asn1.html

Sorry to disappoint you in this festive time, but this approach has the same fundamental flaw as prepared SQL statements do. You have to bind native Ada objects to protocol/serialized/persistent objects forth and back. This

does not work well in practice. In fact, it barely work at all considering the overhead and hazards of type conversions.

A different approach is Ada's representation clauses which describe both objects same. Beyond simple textbook cases that does not work either.

The best practical method so far is using manually written stream attributes. Unfortunately it has shortcomings too:

1. Reuse is limited and composition is unsafe because stream attributes are non-primitive operations.
2. Introspection is almost non-existed. Only tagged types could have it.
3. No support of error handling and versioning. Though it is possible to do manually that is extremely error-prone and totally lacks static verification when the number of test cases is huge to potentially infinite. Even worse, the offending cases do not show up in a normally functioning system. So, when detected, it is always too late.

P.S. Needless to say, the problems 1-3 fully apply to other two methods as well.

P.P.S. And nothing was said about referential and recursive types...

From: Olivier Henley <olivier.henley@gmail.com>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Mon, 31 Dec 2018 08:55:40 -0800

Interesting. I do not grasp the problem in full though...

When you say "Sorry to disappoint you in this festive time", do you mean trying a solution from ASN.1 or only trying at Protobuff?

I think I get why a Protobuff could not cover "complete" transfer of Ada types around, but how does other languages do? (Almost everyone has it) Some of these languages have relatively "complex" type system..?

How do they achieve it? They express any complex types with a limited subset of primitive types(string, int32, etc)?

Can you give a more pragmatic example that exemplifies the limitations in Ada?

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Mon, 31 Dec 2018 18:59:35 +0100

>> When you say "Sorry to disappoint you in this festive time", do you mean trying a solution from ASN.1 or only trying at Protobuff?

Both. They are useless, up to harmful.

> I think I get why a Protobuff could not cover "complete" transfer of Ada types around, but how does other languages do? (Almost everyone has it) Some of these languages have relatively "complex" type system..?

The very concept of a data definition/description language (DDL) is wrong as I tried to explain. It has a very long and sad history in process automation, control, communication (e.g. CORBA), databases (e.g. SQL). Almost everybody and everyone tried it and failed. There are countless protocol describing "languages" around in process automation. I fought with them for decades, wrote several compilers for this mess. One could save huge amount of money and time if there were a law to punish people introducing this stuff... (:-))

> How do they achieve it? They express any complex types with a limited subset of primitive types (string, int32, etc)?

You cannot express a type in a DDL. Data /= Type. Type = data + operations. If you want to express complex typed objects you lose before you start with a DDL. You throw all type semantics overboard.

If you are OK without semantics then there is no need to introduce this mess. Use Ada stream attributes and simply read and write what you want and how you want. It is clean, easy, fast and 100% Ada.

> Can you give a more pragmatic example that exemplifies the limitations in Ada?

Any limitations Ada might have are unrelated to the issue of language impedance: DDL vs Ada unless you make DDL embedded like embedded SQL, which does not work either.

I believe AdaCore has a product of the sort. Though I don't think that would be much better, but I would rather trust them than anybody else...

From: G. B. <nonlegitur@nmhp.invalid>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Wed, 2 Jan 2019 06:57:14 -0000

> *If* you are OK without semantics then there is no need to introduce this mess. Use Ada stream attributes and simply read and write what you want and how you want. It is clean, easy, fast and 100% Ada.

What kind of stream do you write for your partners in business? Three of them have different needs than you WRT data and none of them is using Ada.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Wed, 2 Jan 2019 11:02:10 +0100

> [...]

> What kind of stream do you write for your partners in business?

Stream of octets.

> Three of them have different needs than you WRT data and none of them is using Ada.

They still can read and write the stream. You are confusing description of a protocol with the implementation of.

The OP suggested having descriptions in protobuff and partial implementation generated from that. It is a bad idea.

BTW, it is very easy to write things like protobuff straight in Ada with Simple Components

http://www.dmitry-kazakov.de/ada/components.htm#17.2.1

This feature is rarely used because, as I said, the concept is too limited and constraining if not wrong altogether.

Here is a small example. Consider an example in protobuff:

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

This direct Ada code:

```
type Person is new State_Machine with
  Name  : String_Data_Item
                (Max_String_Length);
  ID    : Unsigned_32_Data_Item;
  Email : String_Data_Item
                (Max_String_Length);
end record;
```

Thanks to Ada's "introspection" that is all. It will be read or written by the connections server automatically. On the packet receipt callback, you get values like Person_Session.ID.Value. Before sending a new packet you assign Person_Session.ID.Value. Note, this is Ada 95, no fancy stuff.

I didn't show here alternation for using optional fields because the transport level representation would be different anyway. Which is the point actually. Such key details are all left unspecified in the protobuff "description" above along with endianness and other encoding issues. Yet exactly these details are essential in practice where the protocol is already defined. Present or not bits might kept combined in the message header, special values of integers are reserved to indicate exceptional states and so on and so forth. And, again, no semantics whatsoever, just buckets of bits.

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Subject: Re: protobuff for Ada
Newsgroups: comp.lang.ada
Date: Tue, 1 Jan 2019 09:05:38 +0100

From my perspective absolutely biggest flaw with technologies like protobuff is:

* Its backed by a large corporation.

* The technology is well known.

* 99.9% of the programming population think that they are the salvation to serialization.

* The licensing is open.

And on top.

* There are significantly more than one project where the lack of protobuff support has ruled out Ada as implementation technology.

And my intent was to eliminate at least the last points even if the technology is inferior.

AdaControl

From: "J-P. Rosen" <rosen@adalog.fr>
Subject: [Ann] AdaControl V1.20r7 released
Newsgroups: comp.lang.ada
Date: Thu, 3 Jan 2019 14:03:30 +0100

Adalog is pleased to announce the release of a new version of AdaControl. Thanks to the support of several sponsors, there are several interesting new controls (see file HISTORY), with a grand total of 70 rules and 565 possible tests! The automatic fixes feature has been extended too.

More details, download, etc. from http://adacontrol.fr. The executable version is now provided for Gnat Community edition 2018.

Reminder: If you have any issue with AdaControl, please report it using

http://sourceforge.net/p/adacontrol/ticket

And if you use it for an industrial project, commercial support is available from Adalog, don't hesitate to ask for information at info@adalog.fr

GNU ELPA

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: GNU ELPA package ada-ref-man version 2012.4 is now available
Newsgroups: comp.lang.ada
Date: Sat, 5 Jan 2019 10:26:23 -0800

GNU ELPA package ada-ref-man version 2012.4 is now available. This version adds '<' '>' annotation to indicate italics in syntax element names:

```
generic_instantiation ::=
   package defining_program_unit_name is
      new <generic_package_>name
         [generic_actual_part]
            [aspect_specification];
```

Simple Components

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada v4.36
Newsgroups: comp.lang.ada
Date: Tue, 8 Jan 2019 12:50:31 +0100

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:

- The package GNAT.Sockets.Server.Blocking was added to provide connection servers handling blocking I/O;

- Procedures Send_Socket and Receive_Socket were added to the package GNAT.Sockets.Server;

- Procedures Reconnect and Request_Disconnect were added to the package GNAT.Sockets.Server;

- The functions Is_Configured, Is_In, Has_Device_Configuration were added GNAT.Sockets.Connection_State_Machine.ELV_MAX_Cube_Client;

- Airing time decoding/encoding error in GNAT.Sockets.Connection_State_Machine.ELV_MAX_Cube_Client.

SparForte

From: koburtch@gmail.com
Subject: Ann: SparForte 2.2
Newsgroups: comp.lang.ada
Date: Tue, 8 Jan 2019 20:15:29 -0800

SparForte version 2.2 was released over the holidays.

It is available for download from the SparForte website:

https://www.sparforte.com/

This version brings preliminary programming-by-contract, side-effect detection and additional shell features. An overview can be found on my blog:

https://www.pegasoft.ca/coder/coder_december_2018.html

There are also several recent blog articles on the design of SparForte, as requested by the mailing list subscribers.

SparForte is a shell, scripting language and web template engine with a core feature set based on Ada. I hope you will find it useful.

Note: I do not regularly read this newsgroup. Please direct questions to the SparForte mailing list.

VTKAda

From: leonid.dulman@gmail.com
Subject: VTKAda 8.2.0
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 11:19:09 -0800

I'm pleased to announce VTKAda version 8.2.0 free edition release 01/02/2019.

VTKAda is Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt5 application and UI framework by Nokia VTK version 8.2.0, Qt version 5.12.0 open source and vtkc.dll, vtkc2.dll, qt5c.dll (libvtkc.so, libvtkc2.so, libqt5c.so) were built with Microsoft Visual Studio 2017 (15.9) in Windows (WIN32) and gcc in Linux x86-64

Package was tested with gnat gpl 2017 ada compiler in Windows 10 64bit, Debian 9.4 x86-64

As a role ADA is used in embedded systems, but with VTKADA(+QTADA) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing and many others thinks.

VTKADA you can be used without QTADA subsystem

Qt5Ada and VTKAda for Windows, Linux (Unix)
https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/

Florist

From: "J-P. Rosen" <rosen@adalog.fr>
Subject: Florist is in Ada !
Newsgroups: comp.lang.ada
Date: Tue, 19 Feb 2019 17:10:08 +0100

See: https://www.carolslaneflorist.com/about-us

(found this while browsing for Florist, the Ada interface to Posix) :-)

OpenGLAda

From: Felix Krause <contact@flyx.org>
Subject: ANN: OpenGLAda 0.7.0
Newsgroups: comp.lang.ada
Date: Sat, 9 Mar 2019 19:18:49 +0100

This release includes some additions to the API, but primarily adds GNAT Community 2018 support. It is also the first release with a Windows installer. This installer includes the optional dependencies (GLFW and Freetype) and installs OpenGLAda on top of an existing GNAT installation.

The dependency on the 3rd party library Strings_Edit has been removed and UTF-8 decoding is now part of the project. This hopefully reduces confusion.

Release and further information is available here:

https://github.com/flyx/OpenGLAda/releases

Ada-related Products

SPARK

From: addaon@gmail.com
Subject: New to Spark, working an example
Newsgroups: comp.lang.ada
Date: Sat, 15 Dec 2018 21:43:50 -0800

Folks, new to this list, so not quite sure on etiquette.

I've been trying to understand Spark-2014 well enough to work through an example, and understand the capabilities and workflow of the tools. The example I chose was an example of floor(lg(n)) for n positive.

Rather than put a long post here, I'll refer to my (long) post at stackoverflow:

https://stackoverflow.com/questions/53752715/proving-floor-log2-in-spark. (If this is bad etiquette here, let me know, and I'll fix -- but it does seem a bit silly to duplicate the content in two locations)

Since SO seems to have a very limited Ada/Spark community, I'm hoping someone here can point me in the right direction. Basically, trying to understand what tools I should be trying to understand at this point. :-) Should I be looking at proving this with just a better understanding of how to write loop invariants; through appropriate lemmas; through an external prover like Coq; or something else?

From: Simon Wright <simon@pushface.org>
Subject: Re: New to Spark, working an example
Newsgroups: comp.lang.ada
Date: Sun, 16 Dec 2018 09:48:17 +0000

I don't think there's anything wrong with trying to attract attention (what gets my goat a bit is people posting the same question in both places at the same time).

I have to confess that I hadn't set up my SO account to watch the tags [spark-2014] or [spark-ada] (why both?), or even [gnat] or [ada2012] - rectified. You would have got more views if you'd included [ada] (but not necessarily any (more) answers :)

Your problems are an indication of why I, as a person who has no access to professional SPARK support, haven't invested any effort to speak of in SPARK (my difficulties were with tasking/time rather than mathematical loops, which tend to be rare in control systems).

That said, it looks to me as though the version of gnatprove in GNAT CE 2018 may not fully understand exponentiation:

util.ads:3:14: medium: postcondition might fail, cannot prove
2 ** Floor_Log2'Result <= X

util.ads:3:16: medium: overflow check might fail
(e.g. when Floor_Log2'Result = 0)

From: Brad Moore <bmoore.ada@gmail.com>
Subject: Re: New to Spark, working an example
Newsgroups: comp.lang.ada
Date: Tue, 18 Dec 2018 18:41:59 -0800

I am by no means a SPARK expert, but I am also interested in exploring SPARK capabilities.

My approach led me to the following solution using just the SPARK 2018 GPL download from Adacore.... (no extra provers were needed here, other than the ones that come with GNAT CE 2018)

As an aside, it appears the version of gnatprove in GNAT CE 2018 does have a pretty good understanding of exponentiation, given that I was able to get the following proven.

```
package Util with SPARK_Mode is
  Max_Log2 : constant := Positive'Size - 1;
  subtype Log_Result is Natural
          range 0 .. Max_Log2;

  function Floor_Log2 (X : Positive) return
          Log_Result with
    Global  => null,
    Depends => (Floor_Log2'Result => X),
    Post    => X >= 2**Floor_Log2'Result
      and then X / 2 < 2**Floor_Log2'Result;
end Util;

pragma Ada_2012;
package body Util with SPARK_Mode is
  function Floor_Log2 (X : Positive) return
          Log_Result is
  begin -- Floor_Log2
    Log_Loop :
    for I in Log_Result loop
      pragma Loop_Invariant
          (for all J in 0 .. I => X >= 2**J);
      pragma Assert
          (X / 2 < 2**Log_Result'Last);
      if X / 2 < 2**I then
        pragma Assert (X >= 2**I);
        pragma Assert (X / 2 < 2**I);
        return I;
      end if;
      pragma Assume(I /= Log_Result'Last);
    end loop Log_Loop;
    return Log_Result'Last;
  end Floor_Log2;
end Util;
```

I technically didn't need to use the Global aspect or the Depends Aspect to prove this function, but I think it is a good idea to provide a more detailed contract using additional SPARK and Ada features, when possible.

The approach I took is to first of all make use of Ada 2012 contracts to constrain the results to only allow valid values. The Log_Result subtype only includes valid result values.

I think this is an important goal in general to eliminate bugs, whether writing code for regular Ada as well as SPARK.

My view is that in general, types such as Integer and Float should not be used since they are types that describe memory storage, not types that describe values of interest in the application domain.

By creating types that more accurately represent the application domain, I believe it makes the job of writing proofs in SPARK much easier, since the prover can reason that the values assigned to such values have specific value ranges and properties.

Another point, is to try to write an implementation that is easier to prove. For that reason, I wrote this is a for loop rather than a while loop, because the compiler can reason statically about how many iterations are performed, and what the values of the loop parameters can be.

The prover was able to prove all the assertions in the implementation.

I had to leave in one assumption, (the pragma assume),

```
pragma Assume(I /= Log_Result'Last);
```

Without that, the prover complains that the post condition,

```
X / 2 < 2**Floor_Log2'Result
```

cannot be proven. It appears that the prover is not able to prove that the loop exited by the return statement, rather than iterating the full loop and exiting the loop without entering the if statement.

However, I think this can be visually inspected and confirmed to be true, since the assert for the if statment,

```
pragma Assert(X / 2 < 2**Log_Result'Last);
```

just prior to the if statement was proven.

It follows that if the assertion is true, then the if statement would have to be entered on the following line, and that the return would exit the loop.

Thus, the reader should be able to visually tell that it is impossible to get by the if statement when I = Log_Result'Last, and thus the pragma Assume is true.

The return at the end of the function should never get executed, as the only way to exit the function is via the return inside the loop.

I didn't need to have the return inside the loop for the purpose of proving the function. I just did that to eliminate the need of extra variable declarations.

Probably the prover could be improved so that such an assume could be eliminated while still proving the overall function.

There may be a way to add additional asserts or pragmas to eliminate the need for the pragma Assume. So far I haven't found any, but perhaps someone else might come up with a way. Otherwise, I'm pretty happy with the solution I ended up with, given that the one assume in the

code can be visually checked easily for correctness.

I am sure that other SPARK solutions exist. I think when it comes to proving something, it is better to start with something simple, and to have in mind choosing an implementation that is easier to prove. This should make it easier to arrive at a proof.

From: Simon Wright <simon@pushface.org>
Subject: Re: New to Spark, working an example
Newsgroups: comp.lang.ada
Date: Wed, 19 Dec 2018 16:58:41 +0000

>> util.ads:3:16: medium: overflow check might fail (e.g. when >> Floor_Log2'Result = 0)

> As an aside, it appears the version of gnatprove in GNAT CE 2018 does have a pretty good understanding of exponentiation, given that I was able to get the following proven.

Apparently so. But the part of gnatprove that gives examples of when the assertion might fail is quite misleading: for example,

util.ads:7:14: medium: postcondition might fail, cannot prove
2 ** Floor_Log2'Result <= X
(e.g. when Floor_Log2'Result = 0 and X = 0) *when X is Positive* !!
and util.adb:19:15: medium: overflow check might fail (e.g. when I = 0)

l.18 for I in 1 .. Log_Result'Last loop

l.19 if 2 ** I > X then

From: Brad Moore <bmoore.ada@gmail.com>
Subject: Re: New to Spark, working an example
Newsgroups: comp.lang.ada
Date: Wed, 19 Dec 2018 20:34:13 -0800

I agree that the error messages are misleading, as I was getting similar messages when I was working on this. While the values "0" mentioned in the error messages were confusing to me, I think the messages were helpful at least in suggesting the sort of tests the prover was trying to prove, which ultimately helped me figure out the assertions that were needed to get this to pass. The values given can be a bit of a red herring sometimes, but I think the underlying test described by the message is more helpful. This is my second problem that I attempted to prove in SPARK, so I didn't know if I would succeed, or know much about how to approach this. It's kind of a rewarding feeling when you get the prover to pass.

One suggestion I have to prove post conditions, is to state the post condition as an assert before returning from the subprogram, and work backwards from there.

References to Publications

Ravenscar References

From: lyttlec <lyttlec@removegmail.com>
Subject: Ravenscar References
Newsgroups: comp.lang.ada
Date: Wed, 16 Jan 2019 12:48:28 -0500

Can anyone suggest a good reference on using the ravenscar profile? In the Ada books I have, it only gets a one or two page mention. A reference with an extended case study would be great.

From: Simon Wright <simon@pushface.org>
Subject: Re: Ravenscar References
Newsgroups: comp.lang.ada
Date: Wed, 16 Jan 2019 18:15:03 +0000

You might find something useful at http://cubesatlab.org e.g. http://www.cubesatlab.org:430/PUBLIC/brandon-chapin-HILT-2016.pdf

From: lyttlec <lyttlec@removegmail.com>
Subject: Re: Ravenscar References
Newsgroups: comp.lang.ada
Date: Fri, 18 Jan 2019 14:18:10 -0500

Thanks all for the links. They are a help. However, I'm looking for something along the lines of porting legacy code to be ravenscar "safe".

As an illustration, consider making Dmitry A Kazakov's code meet Ravenscar. I need to port lots of existing more or less standard components to meet Ravenscar. This is to satisfy some regulatory authorities.

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Subject: Re: Ravenscar References
Newsgroups: comp.lang.ada
Date: Sun, 20 Jan 2019 18:12:11 +0100

I don't know that "port" is a good word for this activity. I once looked at implementing Sandén's FMS problem using Ravenscar. Starting from the requirements, I first had to find a Ravenscar-suitable design. The standard design has a dynamic task per job, and is clearly not possible using Ravenscar. An alternative design using a task per workstation had to be used.

From that choice, Ravenscar drove a proliferation of protected objects and helper tasks. Things that were simple in full Ada became much more complex to meet the restrictions of the profile.

Presumably you would need to apply a similar process to each of the components you need to convert.

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Ravenscar References
Newsgroups: comp.lang.ada
Date: Mon, 21 Jan 2019 17:19:43 -0600

Note that the less strict profile Jorvik, defined in Ada 2020 (and already implemented in GNAT) would simplify this process.

I don't think it is possible to "convert" regular Ada code into Ravenscar (unless, of course, it doesn't use any tasks ;-). You pretty much have to completely rewrite it with Ravenscar in mind. (In this way, it is very much like using SPARK.)

From: "J-P. Rosen" <rosen@adalog.fr>
Subject: Re: Ravenscar References
Newsgroups: comp.lang.ada
Date: Tue, 22 Jan 2019 10:25:08 +0100

I don't fully agree with that statement; it all depends where you start from.

I recently helped one of my clients who wanted to move to Ravenscar. The original structure was all Ada83, communicating with rendezvous.

However, it was already safety critical, therefore based on cyclic, never ending tasks, and limited communications. It was reasonably easy to define patterns for matching the existing structure into Ravenscar patterns.

Ada Inside

Compilation Issues

From: alexander@junivörs.com
Subject: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 03:46:02 -0800

I've read some threads on here regarding the licensing situation of AdaCore's Libre compiler. For my upcoming project, I'm going to need (= very strong desire) to use Ada and I'm also going to need to be able to license the executable produced thereof in any way I desire.

In regards to the aforementioned, I have two questions. I realize I come forth as somewhat paranoid in the upcoming paragraphs (which undoubtedly I am). The licensing situation worries me a great deal.

1. ```As for the compiler build provided by (the GetAdaNow Mac OS X section's link to Sourceforge)[1]; which parts of that GCC build for compiling Ada can you safely use and still be covered by the "GCC Runtime Library Exception"? I can see it states you can use `GNATCOLL` and `XMLAda`. I'm assuming the standard library is included as well. Can you on the other hand use all console commands? `gnat <command>`? `gprbuild`? Or would these inject "non-runtime library exception'd" GPL code into the executable?```2. ```I've been attempting to compile and link some code through the use of the `gcc` command solely, but haven't been successful in doing so. I have, on the other hand, been able to successfully generate an

executable by utilizing the `gnatbind` and `gnatlink` commands consecutively after compiling with `gcc -c <file>`. Is it possible to use only the `gcc` command for the matter, or do you need to also throw in a few calls to the `gnat` commands?

When executing the following commands...

$ gcc -c src/main.adb -o obj/main.o

$ gcc -o main obj/main.o

I wind up with the following error (on the second command, which should be a GCC link):

Undefined symbols for architecture x86_64:

"_main", referenced from:

implicit entry/start for main executable

(maybe you meant: __ada_main)

ld: symbol(s) not found for architecture x86_64

collect2: error: ld returned 1 exit status

A similar error occurs when I attempt to create `.so` libraries manually using the `-shared` compiler switch. With all that being said, is it simply not possible to do these things through solely `gcc`, or am I missing something?```

It may be worth noticing that I've fallen in love with Ada to the utmost degree over the past year. As such, I'm planning on, at the very least, stalking "comp.lang.ada" like some creepy figure. You'll probably see more from me beyond these first two questions, is what I'm saying.

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GCC 20Mac OS X/8.1.0/native-2017/

From: Simon Wright <simon@pushface.org>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 16:11:48

Let me start by saying that I'm not a lawyer.

> 1. ```As for the compiler build provided by (the GetAdaNow Mac OS X section's link to Sourceforge)[1]; which parts of that GCC build for compiling Ada can you safely use and still be covered by the "GCC Runtime Library Exception"? I can see it states you can use `GNATCOLL` and `XMLAda`. I'm assuming the standard library is included as well. Can you on the other hand use all console commands? `gnat <command>`? `gprbuild`? Or would these inject "non-runtime library exception'd" GPL code into the executable?```

They may (do) *generate* source code that gets included in the executable (gnatbind does this). But that isn't code that's provided with the compiler and might have a copyright issue; it's no different in principle from object code generated directly by the compiler.

> 2. ```I've been attempting to compile and link some code through the use of the `gcc` command solely, but haven't been successful in doing so. I have, on the other hand, been able to successfully generate an executable by utilizing the `gnatbind` and `gnatlink` commands consecutively after compiling with `gcc -c <file>`. Is it possible to use only the `gcc` command for the matter, or do you need to also throw in a few calls to the `gnat` commands?

[...]

Building even hello_world* is sufficiently complex that you need gnatbind, gnatlink. As you've seen, you can use gcc for the actual compilation.

Building a dynamic library (do you mean .so? are you on a Mac or Linux?

You mention my darwin 8.1.0 release) is more so.

To see what gnatbind gets up to while doing its work, look at the b__* (or b~*) files it generates. Not much fun or point in generating those by hand.

* You can build a simple null program for an embedded system on an MCU without gnatbind, gnatlink. But you have to bother about storage mappings, prcessor startup, linker scripts etc instead.

From: Lucretia <laguest9000@googlemail.com>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 08:31:59 -0800

[...].

What version is that compiler on sourceforge? Is it from FSF directly, i.e. gcc.gnu.org? Or is it GNAT-GPL/CE, i.e. from AdaCore.com? If the latter, the licence is GPL-3.0 no linking exception, otherwise it's GPL-3.0 with linking exception. Basically, avoid anything GPL-3.0 no linking exception, especially Adacore's libraries.

From: G. B. <nonlegitur@nmhp.invalid>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 18:50:45 -0000

> I've read some threads on here regarding the licensing situation of AdaCore's Libre compiler. For my upcoming project, I'm going to need (= very strong desire) to use Ada and I'm also going to need to be able to license the executable produced thereof in any way I desire.

For licensing in arbitrary ways, the aforementioned Ada distribution is not the suitable one. Another compiler distribution might meet your needs, including some FSF GNAT. GPL means tit-for-tat and thus intentionally puts restrictions on licensing, no back doors.

From: Simon Wright <simon@pushface.org>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 19:21:04 +0000

> What version is that compiler on sourceforge? [...]

It's vanilla FSF with Adacore libraries, some of which have the runtime library exception, some of which don't (as noted at the link).

The Adacore sources, at https://github.com/AdaCore, are on the whole GPLv3 with the runtime exception. I've taken care to report the status:

from https://sourceforge.net/projects/gnuada/files/GNAT_GCC MacOS X/8.1.0/native-2017/

Tools included:

Full GPL:

ASIS from https://github.com/simonjwright/ASIS at [8ba68f3].

AUnit and GDB from GNAT GPL 2017.

Gprbuild from https://github.com/AdaCore/gprbuild at commit [1e551df] (note, libgpr is GPL with Runtime Library Exception[1]).

GPL with Runtime Library Exception[1:

GNATCOLL from:

https://github.com/AdaCore/gnatcoll-core at commit [a093d11].

https://github.com/AdaCore/gnatcoll-bindings at commit [2c426fe].

https://github.com/AdaCore/gnatcoll-db at commit [b66441c].

XMLAda from https://github.com/AdaCore/xmlada at commit [8a4b2bf]

From: alexander@junivörs.com
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 12:50:42 -0800

> Building a dynamic library (do you mean .so? are you on a Mac or Linux?

> You mention my darwin 8.1.0 release) is more so.

Yes. According to (this page)[1] it's accomplishable using the following command:

$ gcc -shared -o libmy_lib.so *.o

but that causes an error mentioning how there are "Undefined symbols for architecture x86_64:".

> For licensing in arbitrary ways, the aforementioned Ada distribution is not the suitable one. Another compiler distribution might meet your needs, including some FSF GNAT. GPL

> means tit-for-tat and thus intentionally puts restrictions on licensing, no back doors.

GPL on its own, I must say, does serve a purpose. It's nice for the author to be able to share their source or works and still be certain nobody can (legally anyway) steal their work and distribute it for a fee themselves.

When it comes to source code licensed under GPL lacking the runtime library exception, on the other hand, I can't say I'm too fond of it. Compilers on their own, featuring a standard library, should always be free to use; whereupon the user may licence their executable in any way they want.

[1] http://beru.univ-brest.fr/~singhoff/DOC/LANG/ADA/gnat_ugn_20.html

From: Simon Wright <simon@pushface.org>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Tue, 11 Dec 2018 23:45:48 +0000

> [1] http://beru.univ-brest.fr/~singhoff/DOC/LANG/ADA/gnat_ugn_20.html

Because that page (and even the latest one at [2]) is wrong.

Almost all Ada code requires the services of the Ada runtime, and you need to reference the runtime at the link stage.

$ gcc -shared -o libmy_lib.dylib *.o -L<whereever> -lgnat -lgnarl

(<whereever>: e.g. /opt/gcc-8.1.0/lib/gcc/x86_64-apple-darwin15/8.1.0/adalib)

This is why it is *so* much easier to use gprbuild (I see that that reference talks about using gnatmake; that's because gnatmake is part of GCC Ada, and gprbuild isn't. But modern gnatmakes will delegate to gprbuild if they find one, at any rate if libraries are involved; they can't generate libraries, because it's too complicated for Adacore to maintain in two places, the GCC tree and the gprbuild tree).

If you want to see what's going on you can use -v.

[2] http://docs.adacore.com/gnat_ugn-docs/html/gnat_ugn/gnat_ugn/the_gnat_compilation_model.html#general-ada-libraries

>> For licensing in arbitrary ways, the aforementioned Ada distribution >> is not the suitable one. Another compiler distribution might meet >> your needs, including some FSF GNAT. GPL means tit-for-tat and thus intentionally puts restrictions on licensing, no back doors.

> GPL on its own, I must say, does serve a purpose. It's nice for the author to be able to share their source or works and still be certain nobody can (legally anyway) steal their work and distribute it for a fee themselves.

> When it comes to source code licensed under GPL lacking the runtime library exception, on the other hand, I can't say I'm too fond of it. Compilers on their own, featuring standard library, should always be free to use; whereupon the user may licence their executable in any way they want.

I don't understand. The first para says it's good, the second says it's bad.

From: alexander@junivörs.com
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Wed, 12 Dec 2018 01:34:01 -0800

> I don't understand. The first para says it's good, the second says it's bad.

Perhaps I've misunderstood something regarding the licensing situation. Is not the reason you cannot use a bunch of AdaCore developed packages due to the fact that it's licensed under GPL without the runtime library exception, ultimately meaning your executable must be licensed under GPL too?

Let's assume someone made a tool to aid people with a repetitive task in Ada. Give that the GPL license and it'd be impossible for someone to "steal" (redistribute for a fee) the original author's code, still allowing people to learn from the code that makes up the tool.

In the second situation, I'm speaking of any library package offering nigh on essential functionality to a programming language (in this case Ada), that does not contain the runtime library exception. I believe that all code developed to ship with a compiler should contain that exception.

I will make sure to await further responses before I justify my belief mentioned in the previous paragraph, should I prove to having gotten something wrong.

Whilst quickly scouring the Internet for some information that would substantiate the claim that some library package files do not contain the runtime library exception, I came across the (`GNAT.Regpat` source)[1], which does contain some form of the runtime library exception.

I presume perhaps that is an older source file than the one shipped with the compiler at this day (Copyright (c) 1996-2002)?

[1] https://www2.adacore.com/gap-static/GNAT_Book/html/rts/g-regpat__adb.htm

From: Björn Lundin <b.f.lundin@gmail.com>
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Thu, 13 Dec 2018 10:21:54 +0100

> [...]

You can always "steal" GPL code, and redistribute it for a fee as you see fit. The freedom in GPL is not free as free beer, but free as free speach. So you would need to provide the sources to the customers you sell to. And I think, a fairly easy way to reproduce an executable/library.

You code depending on GPL (linked with) will inherit the GPL license.

But you can charge your customers whatever you want.

However you likely need to provide something better that the original code for people _wanting_ to pay you, I guess.

From: alexander@junivörs.com
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Thu, 13 Dec 2018 02:30:20 -0800

> [...]

I don't know wherefrom I got my information that you can't sell a GPL application. Thank you for clarifying this!

From: alexander@junivörs.com
Subject: Re: Licensing Paranoia and Manual Compilation Issues
Newsgroups: comp.lang.ada
Date: Thu, 13 Dec 2018 02:32:47 -0800

> I don't know wherefrom I got my information that you can't sell a GPL application. Thank you for clarifying this!

Or rather, clarifying the contrary; correcting me.

Coextension Bug In GNAT

From: Jere <jhb.chat@gmail.com>
Subject: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 07:59:00 -0800

I was messing around and trying to learn coextensions and I came across some counter intuitive functionality. If I directly initialize one via an aggregate, it works fine.

However, if I initialize through a constructing function, it seems to treat the access discriminant as a normal access type and finalizes it at the end of the program instead of when the object leaves scope. I don't fully understand them yet and there isn't much on them listed in the RM but one section (at least according to the index)[1]. That one section does indicate that initialization via a function should be valid however, so maybe I am back to I am doing it wrong or potentially a GNAT bug.

I'm using GNAT 7.1.1

Here is my test program

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Finalization; use Ada.Finalization;

procedure Hello is

   type Thing_1 is new Limited_Controlled
           with null record;

   overriding
   procedure Finalize(Self : in out Thing_1)
   is
   begin
      Put_Line("Finalize Thing_1");
   end Finalize;

   type Thing_2
      (Other : not null access Thing_1)
   is limited null record;

   procedure Test_Coextension_1 is
      The_Thing : Thing_2(new Thing_1);
   begin
      Put_Line("Coextension directly
            initialized");
   end Test_Coextension_1;

   function Make_Thing_2 return Thing_2 is
   begin
      return (Other => new Thing_1);
   end Make_Thing_2;

   procedure Test_Coextension_2 is
      The_Thing : Thing_2 := Make_Thing_2;
   begin
      Put_Line("Coextension initialized
            through build in place");
   end Test_Coextension_2;

begin
   Test_Coextension_1;
   Test_Coextension_2;
   Put_Line("Test Finished");
end Hello;
```

Any thoughts?

[1] Ada 2012 tc1 RM 3.10.2(14.4/3) - http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-3-10-2.html#I2301

From: Jere <jhb.chat@gmail.com>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 08:02:27 -0800

> [...]

Sorry, forgot to put the program output:

$gnatmake -o hello *.adb

gcc -c hello.adb

gnatbind -x hello.ali

gnatlink hello.ali -o hello

$hello

Coextenson directly initialized

Finalize Thing_1

Coextension initialized through build in place

Test Finished

Finalize Thing_1

From: Simon Wright <simon@pushface.org>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 16:56:11 +0000

> [...]

Compiling with -gnatwa I see "warning: coextension will not be finalized when its associated owner is deallocated or finalized", so GNAT clearly meant to do it!

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 20:16:09 -0600

> [...]

This message is nonsense, because a coextension is effectively part of the associated object. What they presumably mean to say is that the declaration in question is *not* a coextension, thus it will not be finalized with the owner.

P.S. I hate coextensions. One of the least necessary complications of Ada.

(Janus/Ada gives you a "feature not implemented" message if you try to create one.)

From: Jere <jhb.chat@gmail.com>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Fri, 21 Dec 2018 03:24:43 -0800

> [...]

> Compiling with -gnatwa I see "warning: coextension will not be finalized when its associated owner is deallocated or finalized", so GNAT clearly meant to do it!

that's pretty interesting. The compiler I was using did not give that warning when compiled with -gnatwa. You're right, that definitely looks intentional.

From: Simon Wright <simon@pushface.org>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 17:58:11 +0000

> [...]

> procedure Test_Coextension_1 is

> The_Thing : Thing_2(new Thing_1);

This is a case of 14.1/3, an allocator used to define the discriminant of an object,

> begin

> Put_Line("Coextension directly initialized");

> end Test_Coextension_1;

> function Make_Thing_2 return Thing_2 is

> begin

> return (Other => new Thing_1);

I think GNAT thinks this is a case of 14.2/3, an allocator used to define the constraint in a subtype_indication, though I'm hard put to it to see the difference from the first case.

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Thu, 20 Dec 2018 20:25:40 -0600

> This is a case of 14.1/3, an allocator used to define the discriminant of an object,

Right, because 14.2/3 says "subtype_indication in any other context", meaning that 14.1/3 applies in an object declaration.

> I think GNAT thinks this is a case of 14.2/3, an allocator used to define the constraint in a subtype_indication, though I'm hard put to it to see the difference from the first case.

That doesn't make any sense, since 14.2/3 is talking about a syntactic subtype_indication, and there is no subtype_indication in the above aggregate. 14.2/3 would be talking about a case like:

```
function Make_Thing_3 return Thing_2 is
   subtype Silly is Thing_2 (new Thing_1);
   Some_Thing : Silly;
begin
   return Some_Thing;
end Make_Thing_3;
```

This function does NOT define a coextension.

So it does look like a GNAT bug. There is the possibility that they are associating the discriminant with the temporary object associated with the allocator, and not the return object, but that seems unnecessarily unfriendly of an interpretation. And it would be wrong for any type that requires built-in-place (I didn't look at the actual declaration of the type). I think the rules are supposed to prevent that interpretation, but whether they really do is an interesting question that I have no interest in exploring.

P.S. Did I mention I hate coextensions?? They provide an endless opportunity to puzzle over rules that really don't matter in the end (and most likely aren't quite right). I suppose they've helped me keep employed running the ARG. :-)

From: Jere <jhb.chat@gmail.com>
Subject: Re: Potential Coextension Bug in GNAT
Newsgroups: comp.lang.ada
Date: Fri, 21 Dec 2018 03:32:03 -0800

> So it does look like a GNAT bug. There is the possibility that they are associating the discriminant with the

> temporary object associated with the allocator, and not the return object, but that seems unnecessarily unfriendly of an interpretation. And it would be wrong for any type that requires built-in-place (I didn't look at the actual declaration of the type). I think the rules are supposed to prevent that interpretation, but whether they really do is an interesting question that I have no interest in exploring.

Ok, that makes me feel better. I was concerned I was misinterpreting the RM about the function return (for build in place). The type was limited, which I believe requires it to be built in place.

> P.S. Did I mention I hate coextensions?? They provide an endless opportunity to puzzle over rules that really don't matter in the end (and most likely aren't quite right). I suppose they've helped me keep employed running the ARG. :-)

Overall, they aren't super useful and are not very intuitive. I don't know the history for why they were added to the language though. I will say they do provide one thing to Ada that no other feature in the language seems to, so there is that. But I don't know the cost versus reward of them.

grpexec Tool

From: VM Celier <vmcelier@gmail.com>
Subject: New tool "gprexec", basically "make with project file"
Newsgroups: comp.lang.ada
Date: Fri, 11 Jan 2019 14:00:10 -0800

I am starting a new project that I have been thinking for several years: gprexec.

gprexec is a "Make build automation tool using GPR project files to describe goals, dependencies, and processes".

It uses a new package: Execution.

Here is an example of a project that can be used by gprexec:

```
project Toto is
  for Main use ("toto.adb");
  package Execution is
    for Process ("display_main") use ("cat",
          "toto.adb");
    for Dependency ("display") use
          ("display_main");
    for Process ("display") use ("cat",
          "toto.gpr");
    for Process ("date") use ("date");
    for Process ("toto") use ("gprbuild", "-f",
          "-q", "toto.gpr");
    for Dependency ("default") use
          ("display", "toto", "date");
    for Process ("default") use ("toto");
  end Execution;
end Toto;
```

Package Execution has these attributes:

- Dependency, to indicate the goals that need to be processed before the indexed goal.
- Process, to indicate the process to be invoked, with its arguments, for the indexed goal.

gprexec needs to be invoked with a single project file and an optional goal. When no goal is specified on the command line, the goal "default" is implied.

For example with the project file toto.gpr above, invoking

gprexec toto.gpr

the goal default will be used, and according to the dependencies processes will be invoked in the following order:

(goal "display_main): cat toto.adb

(goal "display"): cat toto.gpr

(goal "toto"): gprbuild -f -q toto.gpr

(goal "date"): date

(goal "default"): toto

After displaying the main toto.adb and the project file toto.gpr, toto.adb is compiled, bound and linked, the date is displayed and the executable "toto" is invoked.

gprexec uses the project file "gpr.gpr", part of the gprbuild repository.

I just created a public repository for gprexec on Github:

https://github.com/vmcelier/gprexec

Anybody interested?

-- Vincent Celier

(no longer associated with AdaCore)

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: New tool "gprexec", basically "make with project file"
Newsgroups: comp.lang.ada
Date: Mon, 14 Jan 2019 13:06:35 -0800

> [...]

Yes, but no.

Some of the ideas behind GPR are good, but if we're being honest its tendency to be "stringly-typed" is annoying given its obvious designed similarity to Ada -- and there are a lot of missed opportunities -- and the sort-of configuration purposes which don't fully support producing an Ada executable (e.g. IIRC you have to use a completely separate configuration to handle DSA.)

From: VM Celier <vmcelier@gmail.com>
Subject: Re: New tool "gprexec", basically "make with project file"
Newsgroups: comp.lang.ada
Date: Mon, 14 Jan 2019 16:49:14 -0800

> Some of the ideas behind GPR are good, but if we're being honest its tendency to be "stringly-typed" is annoying given its obvious designed similarity to Ada

It is true that the syntax of the project language is similar to the one of Ada, but there is a big difference between the two languages:

- Ada is an executable language
- the project language is a declarative language

You don't "execute" project files, you use it to describe a system for different tools. This is why there are almost no types in the project language because types are not really needed and they would complexify the language for no real benefit.

> -- and there are a lot of missed opportunities

Could you tell us one or two of these missed opportunities?

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: New tool "gprexec", basically "make with project file"
Newsgroups: comp.lang.ada
Date: Tue, 15 Jan 2019 08:41:01 -0800

> It is true that the syntax of the project language is similar to the one of Ada, but there is a big difference between the two languages:

> - Ada is an executable language

> - the project language is a declarative language

This is actually less of an issue than might be thought; though some of the "fix-ups" might be a bit stifling to some. You could, for example, impose restrictions/mandatory-structure on the configuration and have all configurations be valid Ada.

> You don't "execute" project files, you use it to describe a system for different tools. This is why there are almost no types in the project language because types are not really needed and they would complexify the language for no real benefit.

No, real enumerations (and attendant Ada-like case-coverage) would be excellent for providing bounded alternations of the configuration.

> > -- and there are a lot of missed opportunities

> Could you tell us one or two of these missed opportunities?

Given Ada's strong generic-system configurations could be described as generic parameters [esp enumerations], which the tools could use to provide bounded options in the absence of defaults.

Package PROJECT_NAME

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: New tool "gprexec", basically "make with project file"
Newsgroups: comp.lang.ada
Date: Tue, 15 Jan 2019 09:22:07 -0800

Sorry, I accidentally submitted the form while composing my example... which is here:

```
Package PROJECT_NAME is
  Type Archetectures is ( x86, x86_64, ARM,
          SPARC, MIPS_V );
  Type Node_Type is (Storage, Processing);
  Type Partition_Type is (Active, Passive);
  Type Compilation_Parameters is record
     CPUs : Natural := 0; -- Use as many
                          -- cores as available.
     Symbols : Boolean := True; -- Don't strip
                                -- symbols.
     Target  : Archetectures;
     --...
  end record;

  Type Partition( Params :
          Compilation_Parameters; Style :
          Partition_Type ) is record
    null; --... Other DSA parameters.
  end record;

  Type Node( Style : Node_type ) is record
    Archetecture : Archetectures;
    case Style is
           when Storage =>    null; --...
           when Processing => null; --...
          end case;
  end record;

  Generic
    Params : Compilation_Parameters;
  Procedure Compile;

  --- CONCEPTUAL GENERIC PACKAGE
  Generic
    Partitions : Array (Positive range <>) of
not null access Partition;
  Package Compiler is
    Procedure Execute;
  End Compiler;

  --- CONCEPTUAL BODY FOR COMPILER
  Package Body Compiler is
    Procedure Execute is
          Begin
            For P of Partitions loop
              declare
                    Procedure Make is new
                    Compile( P.Params);
                    begin
                      Make;
                    end;
            End loop;
          End Execute;
  End Compiler;

End PROJECT_NAME;
```

Now, obviously there would have to be standardization -- and it would probably work better if "Archetectures" were a parameter to PROJECT_NAME -- because if all config-packages were generic we could "nest" dependencies:

```
Generic
  Type STANDARD_PARAM is limited
          private;
  -- "Configuration standard param"
  with Package P1 is new Project_1
          (STANDARD_PARAM );
  with Package P2 is new Project_2
          (STANDARD_PARAM );
  -- P3 depends on P1&2
  with Package P3 is new Project_3
          (STANDARD_PARAM, P1, P2 );
Package Project_4 is
  -- ... STANDARD STRUCTURE.
End Project_4;
```

Now, all of that is operating with the idea of using Ada as a config-language, which is doable, but perhaps a bit ugly... It might be a bit better to sit down, think about configurations (esp. in the presence of DSA) and develop an Ada-like language for that. (Perhaps in conjunction with a new Ada IR similar to DIANA, such that this configuration-description "compiles down to" the proper generic-nodes which can then be interpreted by the compiler as the configuration[s] to use; or processed by tools to inter-operate with current tools [ie IR → (GPR_File, Gnatdist_Configuration_File) for GNAT].)

Program entry in GPR

From: Jesper Quorning <jesper.quorning@gmail.com>
Subject: Package procedure as program entry in GPR project
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 07:12:22 -0800

Hello All,

With the package specification:

```
package My_Program_Package is
  procedure Program_Entry_Procedure;
end My_Program_Package;
```

How do i make Program_Entry_Procedure as the program entry procedure in a GPR project?

I think it is possible, but cannot find out how.

I know how to use a stand-alone procedure file as program entry and how to name the executable.

From: Jere <jhb.chat@gmail.com>
Subject: Re: Package procedure as program entry in GPR project
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 09:05:24 -0800

> [...]

With that specific setup, I am not sure. But if you are willing to change a couple of things you can do:

```
-- my_program_package.ads
package My_Program_Package is
  -- Notice no declaration here for the
  -- procedure, but you can put other
  -- things if you like
end My_Program_Package;

-- my_program_package-
program_entry_procedure.adb
procedure My_Program_Package.
          Program_Entry_Procedure is
begin
  -- your main stuff
end My_Program_Package.
Program_Entry_Procedure;
```

Then you modify the GPR file to point to it as the main:

```
for Main use ("my_program_package-
program_entry_procedure.adb");
```

I do something similar for my Gnoga GUI projects so I can have program level stuff in the top package but have the main a child of that top level package.

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Package procedure as program entry in GPR project
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 15:42:12 -0600

> [...]

I realize you are asking for GPR, so by definition you don't care about portability, but:

Ada only requires Ada implementations to support library-level procedures as the main. See 10.2(29). A particular implementation can allow more, but there is no requirement.

So if you ever might want to use some other Ada compiler (I for one, hope so), use such a routine.

It's trivial to write one, after all:

```
with My_Program_Package;
procedure My_Program_Main is
begin
    My_Program_Package.
        Program_Entry_Procedure;
end My_Program_Main;
```

From: Jesper Quorning <jesper.quorning@gmail.com>
Subject: Re: Package procedure as program entry in GPR project
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 17:47:30 -0800

I just wanted a way to avoid the trivial main file.

I also considered

```
package simple is
  procedure main
end simple;

package body simple is
  procedure main is
  begin
    ...
  end main;
private
  main;
end simple;
```

But GPR would not do that either. I will stick to the simple procedure file.

From: Simon Wright <simon@pushface.org>
Subject: Re: Package procedure as program entry in GPR project
Newsgroups: comp.lang.ada
Date: Sat, 26 Jan 2019 12:05:35 +0000

> [...]

This isn't a GPR thing, it's a GNAT thing: GNAT has no extensions here beyond the requirement.

If you have a minimal bare-board project without any requirement for the Ada runtime system, it's possible to do what you ask: see Maciej Sobczak's 'Ada and SPARK on ARM Cortex-M' tutorial[1], in particular the 'First Chapter'[2].

It would be hard (and pointless) to attempt this for a program intended to run on a typical operating system.

[1] http://www.inspirel.com/articles/Ada_On_Cortex.html

[2] http://www.inspirel.com/articles/Ada_On_Cortex_First_Program.html

GNAT Bug

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 06:51:50 -0800

This will probably sound more like venting frustration. Sorry if so. But how does anybody get anything done? gnat is *the major* Ada compiler and pretty much the only one implementing the standard in full. Yet I cannot seem to get it working past really small size in any project. As soon as I try to get any basic type composition done (only 3-4 inheritance levels, with, perhaps double interface overlay), I get that dreaded gnat bug message.. This is at least the 3rd one just within past week or two..

Specifically this:

https://github.com/gerr135/wann/tree/gnat_bug01

(the bug triggering code is in a separate branch pointed to by that link).

This is still early in design phase and far from being functional in any way, so I don't really expect much comments on the code itself (thus that "venting frustration" comment above). But the pattern that seems to universally trigger these gnat bugs is something along these lines:

```
type Base_Interface is interface;
..

type Derived1_Interface is new Base_Interface and ..;
..
```

perhaps few more layers here..

then

```
type Base_impl1 is new Base_Interface with private;
..
type Derived1 is new Base_impl1 and Derived1_Interface with private..
```

basically trying to stitch together functional interface hierarchy (containing algorithmic stuff) and data storage type hierarchy. Somehow gnat very often just cannot handle this type of design :(.

(and yes, I am avoiding having to lay generics on top of other generics like Dmitry suggests - keeps design and compilation times sane, but apparently overloads gnat capacity to deal with abstraction).

So, I guess my question would be - how people deal with such situations (combining algorithmic and data representation type hierarchies) in their experience? Or, whether too many child modules makes any difference? I seem to have noticed that the more hierarchical my packages are (but this one is only like 3rd level child!) the more often I trigger that gnat bug message.. (but then keeping the code in one huge module is really messy too!)

And yeah, the specific message here is:

```
gprbuild -P wann.gpr
Compile
   [Ada]          run_customnn.adb
+===GNAT BUG DETECTE===+
| Community 2018 (20180524-73) (x86_64-pc-linux-gnu) GCC error: |
| in gnat_to_gnu_entity, at ada/gcc-interface/decl.c:429 |
| Error detected at wann-nets-vectors.ads:106:5 [run_customnn.adb:23:5]    |
| Please submit a bug report by email to report@adacore.com.  |
| GAP members can alternatively use GNAT Tracker:   |
| http://www.adacore.com/ section 'send a report'.  |
| See gnatinfo.txt for full info on procedure for submitting bugs.         |
| Use a subject line meaningful to you and us to track the bug.            |
| Include the entire contents of this bug box in the report.               |
| Include the exact command that you entered.                              |
| Also include sources listed below.
| Use plain ASCII or MIME attachment(s).                                   |
+=========================+
```

and the "please include" list of files lists pretty much all of them in the src dir.

But as I said, this is rather a pattern I observe, not just one-off situation..

This is with the latest FSF gnat compiler (2018 release based on gcc-7.3.1 backend, Gentoo Linux, relatively fresh everything else).

Sigh, I guess another report to file with AdaCore..

Sorry for disturbance here..

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 19:47:31 +0100

> So, I guess my question would be - how people deal with such situations (combining algorithmic and data representation type hierarchies) in their experience? Or, whether too many child modules makes any difference? I seem to have noticed that the more hierarchical my packages are (but this one is only like 3rd level child!) the more often I trigger that gnat bug message.

Do not panic. In many cases the bug is triggered by an illegal program. Try an older version of GNAT compiler to find what triggers it. In other cases you can work around it using minor code variations.

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 13:32:52 -0800

> [...]

Oh, I am far from panic. It is, as I mentioned, already like 3rd project where I trigger a similar bug in the space of a week or two. Just, when you finally laid out thing just the way you wanted and then gnat explodes on that final compile attempt. Then you get such an expression of frustration :).

Thanks for the advice though! This is pretty much how I handle these. But nice to know I am not alone in this. Well, in fact not so nice - would be nicer if this never happened of course :). But at least reassuring. So thanks again.

From: Simon Wright <simon@pushface.org>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 01 Feb 2019 20:41:06 +0000

> gprbuild -P wann.gpr
>
> Compile
>
>   [Ada]   run_customnn.adb
>
> +===GNAT BUG DETECTE===+
>
> | Community 2018 (20180524-73) (x86_64-pc-linux-gnu) GCC error: |
>
> | in gnat_to_gnu_entity, at ada/gcc-interface/decl.c:429 |
>
> | Error detected at wann-nets-vectors.ads:106:5 [run_customnn.adb:23:5] |

but I get

```
$ gprbuild -p -P wann
wann.gpr:5:32: "../../libs/ada_common/src" is not a valid directory
gprbuild: "wann" processing failed
```

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 13:26:27 -0800

Oops, that's a stale import of an extra lib I thought to use at one point but then rolled back. Apparently I forgot to remove the path, and I obviously still have that lib on my system, even if it is not withed any more.

Removed, you should be able to proceed now. Sorry about that.

One other note: at first build the compiler may complain about missing obj/dbg dir. Please just run:

mkdir -p obj/dbg

from the project dir (not src, one level above it).

I have obj/ in .gitignore to prevent it tracking generated files (and git tends to ignore the entire dir, not just its contents. At least my very short attempts to force it to ignore obj/* but not obj/ itself did not succeed. I preferred the annoyance of running once the mkdir command over spending more time trying to beat git when I set it up).

Thanks for your attempt!

From: Simon Wright <simon@pushface.org>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 01 Feb 2019 23:17:17 +0000

OK, and all the compilers I have here fail in the same way:

FSF GCC 6, 7, 8, 9

GNAT 2016, 2017, 2018

For GCC 9, the relevant code in decl.c is

```
  /* If we get here, it means we have not yet done anything with this entity. If we are not defining it, it must be a type or an entity that is defined   elsewhere or externally, otherwise we should have defined it already. */

  gcc_assert (definition
      || type_annotate_only
      || is_type
      || kind == E_Discriminant
      || kind == E_Component
      || kind == E_Label
      || (kind == E_Constant &&
          Present (Full_View (gnat_entity))
              || Is_Public (gnat_entity));
```

... and we are none the wiser.

I tried

```
  gprbuild -p -P wann.gpr -c -u -f wann-nets-vectors.adb
```

and it compiled OK except for loads of 'unimplemented' warnings.

Poking around at your main program, it seems that things go wrong at the line

```
    package PNetV  is new PNet.vectors;
```

(i.e., I deleted stuff starting at the bottom, by the time I'd deleted this line it compiled "OK".

> One other note: at first build the compiler may complain about missing

> obj/dbg dir. Please just run:

> mkdir -p obj/dbg

> from the project dir (not src, one level above it).

'gprbuild -p' will create missing directories.

Or you could add

```
  for Create_Missing_Dirs use "true";
```

to your GPR (recent ones only).

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Fri, 1 Feb 2019 23:16:32 -0800

> [...]

> I tried

> gprbuild -p -P wann.gpr -c -u -f wann-nets-vectors.adb

> and it compiled OK except for loads of 'unimplemented' warnings.

Ok, so the file itself compiles (I gotta read up on all those switches apparently. This is a ways to quickly test stuff. Thanks for a suggestion!)

But that is quite what I expect, given the nature of the bugs I get - they clearly come from gnat getting lost in all the inheritances I throw at it.

> Poking around at your main program, it seems that things go wrong at the line

The specific offending lines are:

wann-nets-vectors.ads:104 and 106

these two full type definitions (if I comment out one it still fails on the other):

```
   type Cached_Proto_NNet is abstract new Proto_NNet and Cached_NNet_Interface with null record;


   type Cached_Checked_Proto_NNet is abstract new Proto_NNet and Cached_Checked_NNet_Interface with null record;
```

These are null record at the moment, as I did not yet get around to properly implement them. Just placeholders essentially. And this is what might be confusing gnat I suspect. I did not yet try to add any actual data inside.

> 'gprbuild -p' will create missing directories.

> Or you could add

Thanks, I'll add this too.

A small note: I will be at the Fosdem most of today and possibly tomorrow. So, I may not be able to reply in a timely manner these two days.

(But I will surely pass by the Ada dev room today!)

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Sat, 2 Feb 2019 08:13:02 +0100

I did put some effort to reduce the problem and the workaround is quite simple, in file "wann-nets.ads:69" mark the procedure Del_Neuron as abstract instead of null.

Here is the small reproducible I ended up with after stripping the code:

```
pragma Warnings (Off);
generic
   type Real is digits <>;
package wann is
end Wann;
--
generic
package Wann.Neurons is
end Wann.Neurons;
---
generic
package Wann.Nets is
   type NNet_Interface is limited interface;
   procedure Del (Net : in out
            NNet_Interface) is null;
   -- Fails
   -- procedure Del (Net : in out
   -- NNet_Interface) is abstract;-- Works
   type Cached_NNet_Interface is limited
interface and NNet_Interface
end Wann.Nets;
--
generic
package wann.nets.vectors is
   type Proto_NNet is abstract new
            NNet_Interface with NULL record;
   type Cached_Proto_NNet is abstract new
      Proto_NNet and
    Cached_NNet_Interface with null record;
end wann.nets.vectors;
--
pragma Warnings (Off);
with wann.nets.vectors;
procedure run_customNN is
   package PW is new wann(Real => Float);
   package PNet   is new PW.nets;
   package PNetV  is new PNet.vectors;
begin
  null;
end Run_CustomNN;
```

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Sat, 2 Feb 2019 11:05:19 -0800

Wow, thank you for your time!

Looking at how that final code is so small and basic, and that snippet of gnat internals that was dug out on another comment above, it looks like gnat does

not implement null primitives in full.. (which is a pity, as null method makes more sense there than abstract, but well..)

Once I am completely back from Fosdem I'll play with this a bit more, to see if that's package hierarchy, generics or combination thereof that is triggering it and submit a bug with final details.

Thanks again!

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Sat, 2 Feb 2019 22:37:01 +0100

Well I think it's more about deeply nested generics, since that is a real nightmare to implement in its full context.

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Mon, 4 Feb 2019 04:28:45 -0800

Not exactly as far as I can tell.

I have played some more with the code and could simplify it even more - there is no need for that extra top package level. Same thing happens if the interfaces are declared at the top, and overridden in a child. Flat package structure (still generic) compiles fine. Removing generics (and instead doing "type Real is new Float" at the top) given unstable behavior - one time I got the same bug triggered, but after I renamed sources (originally names "workaround" to "alternative" to reflect better the situation) gnat started to compile it properly (giving error message about declaring vars of abstract type). Apparently it has a sense of humor - this is literally the situation of "what is written here is a lie").

Anyway, I have created a github project to keep the code producing gnat bugs I have so far encountered (only one at the moment, but there are two more I need to clean-up and report). This project shows the code triggering the bug, as well as workarounds and the status of the bug report. I think such a resource would be rather useful (given that AdaCore themselves don't really support the bug tracker, at least for the community version [1]). So, please feel free to consult or even contribute, if there are any more commonly encountered bugs.

The project can be found here:

https://github.com/gerr135/gnat_bugs

[1] I chatted with them briefly 2 days ago on Fosdem and they told me that they prefer an email report and that tracker is not really functional for a community version at least.

From: joakimds@kth.se
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Mon, 4 Feb 2019 07:30:30 -0800

George, thanks for your efforts in making detailed gnat bug reports and your input in the Ada dev room on Fosdem 2019.

From: Simon Wright <simon@pushface.org>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Mon, 04 Feb 2019 16:11:47 +0000

> I chatted with them briefly 2 days ago on FOSDEM and they told me that they prefer an email report and that tracker is not really functional for a community version at least.

Do you mean the GCC Bugzilla? I can quite understand why reports against just GNAT CE wouldn't really be appropriate there.

AdaCore do respond to reports on FSF GCC there, especially if the report is about the GCC build system or about bad code generation. However, old bugs don't really get curated as they are fixed in new releases.

This doesn't work where the sources concerned aren't publicly visible in the repository: for example, the embedded runtimes.

Personally I like to report on Bugzilla where appropriate, because reports to report@adacore.com aren't publicly visible. I don't know how annoying it'd be to report in both places.

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Tue, 5 Feb 2019 11:16:51 -0800

> Do you mean the GCC Bugzilla? I can quite understand why reports against just GNAT CE wouldn't really be appropriate there.

No, I meant the tracker mentioned on the bug message:

>GAP members can alternatively use GNAT Tracker: |

>| http://www.adacore.com/ section 'send a report'.

From his reaction I took it that that tracker is not that active. Although it would not be so useful for many people anyway, if it has usage limitations.

> AdaCore do respond to reports on FSF GCC there, especially if the report is about the GCC build system or about bad code generation.

Oh, they do? Thanks for the info!

That's not something I directly thought about, as the problem is with the upstream (of FSF), so it makes sense to take it directly to upstream (the most common reaction of many projects and distributions is to first try to figure out if its them or upstream, and if its upstream, then its universally - "report it to upstream". Which is totally logical, in avoiding messy duplication of effort. In fact it is often not something they would even have control over).

So, I just took it directly to upstream, strictly following the procedure described in the bug message :).

> Personally I like to report on Bugzilla where appropriate, because reports to report@adacore.com aren't publicly visible. I don't know how annoying it'd be to report in both places.

Yes, that's indeed a concern. This is why I created that github project, as I had a few bugs lying around already. I'll populate it with more when I get around to it.

But to the credit of AdaCore, they react quickly - I already got a confirmation that they got it and will look into it..

From: Simon Wright <simon@pushface.org>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Tue, 05 Feb 2019 20:37:16 +0000

> [...]

> From his reaction I took it that that tracker is not that active. Although it would not be so useful for many people anyway, if it has usage limitations.

If you have a contract with AdaCore then Tracker is the point of contact; and the response when I worked for a company with a contract was terrific.

If not, your only direct contact is report@adacore.com (with GNAT in the subject line).

> That's not something I directly thought about, as the problem is with the upstream (of FSF), so it makes sense to take it directly to upstream (the most common reaction of many projects and distributions is to first try to figure out if its them or upstream, and if its upstream, then its universally - "report it to upstream". Which is totally logical, in avoiding messy duplication of effort. In fact it is often not something they would even have control over). So, I just took it directly to upstream, strictly following the procedure described in the bug message :).

The AdaCore people working on FSF GCC are the same people working on the 'upstream' product, which is why I've never thought of it like that; but

I see your point.

And, I've occasionally added 'same problem with GNAT CE' to Bugzilla reports where I thought it might stimulate interest.

> [...]

> But to the credit of AdaCore, they react quickly - I already got a confirmation that they got it and will look into itIt helps if they know you!

From: George Shapovalov <gshapovalov@gmail.com>
Subject: Re: Yet another gnat bug
Newsgroups: comp.lang.ada
Date: Wed, 6 Feb 2019 02:53:18 -0800

> The AdaCore people working on FSF GCC are the same people working on the 'upstream' product, which is why I've never thought of it like that; but

> I see your point.

Oh, so they do have people working on gcc directly? Nice!

Sure, that makes total sense (for a company that essentially sells a gcc-based compiler). But unfortunately this rarely happens in reality.

AdaCore seems like a real nice company! (A bit of praise never hearts, but seriously, thanks to AdaCore people for nice work overall!)

> > But to the credit of AdaCore, they react quickly - I already got a confirmation that they got it and will look into it.

>

> It helps if they know you!

Maybe, but then I only saw them once in a person, and that likely were other people.

But more importantly, this particular issue seems to be a general omission affecting gnat universally, which would affect all kinds of users. I am just puzzled how this thing was not triggered before by at least some users? Is nobody fond of trying to lay out their types in the most abstract way possible? That *does* force better design and ends up saving quite a bit of work down the road (to the point of coding becoming really boring after the general structure is in and successfully compiled by gnat). Well, I guess people just always write "is abstract" even where "is null" would make more sense (or that not many people mix generics and OOP abstraction)..

Alignment issue

From: Simon Wright <simon@pushface.org>
Subject: Alignment issue
Newsgroups: comp.lang.ada
Date: Sat, 16 Feb 2019 19:40:38 +0000

I have code like this (written while working on a StackOverflow question), and GNAT ignores apparent alignment requests.

```
with System.Storage_Pools;
with System.Storage_Elements;
package Alignment_Issue is

   type Data_Store is new
System.Storage_Elements.Storage_Array
   with Alignment => 16;  --
Standard'Maximum_Alignment;

   type User_Pool (Size :
System.Storage_Elements.Storage_Count)
      is  record
        Flag        : Boolean;
        Data        : Data_Store (1 .. Size);
      end record
   with Alignment => 16;  --
Standard'Maximum_Alignment;

end Alignment_Issue;
```

(Standard'Maximum_Alignment is a GNAT special) and compiling with GNAT CE 2018 (and other GNAT compilers) I see

```
$ /opt/gnat-ce-2018/bin/gnatmake -c -u
-f -gnatR alignment_issue.ads

gcc -c -gnatR alignment_issue.ads

Representation information for unit
Alignment_Issue (spec)
for Data_Store'Alignment use 16;
for Data_Store'Component_Size use 8;

for User_Pool'Object_Size use ??;
for User_Pool'Value_Size use ??;
for User_Pool'Alignment use 16;
for User_Pool use record
   Size at 0 range  0 .. 63;
   Flag at 8 range  0 ..  7;
   Data at 9 range  0 .. ??;
end record;
```

which means that GNAT has ignored the alignment specified for Data_Store when setting up User_Pool.Data.

Is this expected? OK?

I found a workround of sorts:

```
type Data_Store (Size :
System.Storage_Elements.Storage_Count)
is record
   Data :
System.Storage_Elements.Storage_Array (1
.. Size);
end record
with Alignment => 16;  --
Standard'Maximum_Alignment;

type User_Pool (Size :
System.Storage_Elements.Storage_Count)
   is record
     Flag  : Boolean;
     Stack : Data_Store (Size);
   end record;
```

giving

```
Representation information for unit
Alignment_Issue (spec)
for Data_Store'Object_Size use ??;
for Data_Store'Value_Size use ??;
for Data_Store'Alignment use 16;
for Data_Store use record
   Size at 0 range  0 .. 63;
   Data at 8 range  0 .. ??;
end record;

for User_Pool'Object_Size use ??;
for User_Pool'Value_Size use ??;
for User_Pool'Alignment use 16;
for User_Pool use record
   Size  at  0 range  0 .. 63;
   Flag  at  8 range  0 ..  7;
   Stack at 16 range  0 .. ??;
end record;
```

(but even then I see that Stack.Data is offset by 8 bytes because of the discriminant)

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Alignment issue
Newsgroups: comp.lang.ada
Date: Mon, 18 Feb 2019 17:01:02 -0600

>I have code like this (written while working on a StackOverflow question), and GNAT ignores apparent alignment requests.

I wouldn't have expected Alignment to cause the effect, but when you specify representation for a record type, any requirements on the components are can be ignored. Perhaps GNAT is taking that somewhat too far??

Ada in Context

Create Attributes

From: eduardsapotski@gmail.com
Subject: Create attributes.
Newsgroups: comp.lang.ada
Date: Fri, 21 Dec 2018 21:37:12 -0800

Sorry for the stupid question...

For example. I have type:

```
type Person is record
   First_Name : Unbounded_String :=
         Null_Unbounded_String;
   Last_Name : Unbounded_String :=
         Null_Unbounded_String;
end record;
```

There is a list:

```
package People_Package is new
Ada.Containers.Vectors(Natural, Person);
People : People_Package.Vector;
```

Next, I want to display this list with headers:

```
----------------------------
|   NAME     |   SURNAME     |
----------------------------
|   John     |    Smith      |
|   Ada      |   Lovelace    |
...
----------------------------
```

Can I use attributes to display headers?

For example something like this:

People'First_Name_Header

How can this be implemented?

From: Brad Moore <bmoore.ada@gmail.com>
Subject: Re: Create attributes.
Newsgroups: comp.lang.ada
Date: Sat, 22 Dec 2018 11:13:40 -0800

You could use a class-wide type or a type with discriminants such as;

```
   type Person_Attribute_Kinds is (Name,
Surname);
     type Person_Attribute (Attribute_Name :
Person_Attribute_Kinds
                    :=
Person_Attribute_Kinds'First) is
     record
       case Attribute_Name is
         when Name | Surname =>
           Name_String : Unbounded_String
:= Null_Unbounded_String;
       end case;
     end record;

     type Person is
       record
         First_Name :
Person_Attribute(Name);
         Last_Name  :
Person_Attribute(Surname);
       end record;

   X : Person;
begin
   Put_Line ("| " &
X.First_Name.Attribute_Name'Image &
           " | " &
X.Last_Name.Attribute_Name'Image & " |");
```

Overloading operators

From: daicrkk@googlemail.com
Subject: Overloading operator “=” for anonymous access types?
Newsgroups: comp.lang.ada
Date: Fri, 11 Jan 2019 13:46:22 -0800

I am working my way through Barnes' excellent Ada book. This is a code sample for deep comparison of linked lists from section 11.7:

```
type Cell is
  record
    Next: access Cell;
    Value: Integer;
  end record;
function "=" (L, R: access Cell) return
Boolean is
begin
  if L = null or R = null then    -- universal =
    return L = R;                 -- universal = (Line
A)
  elsif L.Value = R.Value then
    return L.Next = R.Next;       -- recurses OK
(Line B)
  else
    return False;
  end if;
end "=";
```

I can't seem to wrap my head around why in Line A operator "=" of the universal_access type is called (because of the preference rule), on Line B, however, the user-defined operator "=" is called (which makes recursion possible in the first place), this time with no preference for operator "=" of universal_access.

Both L and R, as well as L.Next and R.Next are of the same anonymous type "access Cell". Why the difference in "dispatching"? Does it have to do with L and R being access parameters? If so, what is the rule there?

I did my best to find anything in the AARM, especially section 4.5.2, but could not make any sense of it.

From: Simon Wright <simon@pushface.org>
Subject: Re: Overloading operator “=” for anonymous access types?
Newsgroups: comp.lang.ada
Date: Sat, 12 Jan 2019 09:50:14 +0000

Given ARM 4.5.2(9.1 ff),

> At least one of the operands of an equality operator for universal_access shall be of a specific anonymous access type. Unless the predefined equality operator is identified using an expanded name with prefix denoting the package Standard, neither operand shall be of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator such that:
>
> * its result type is Boolean;
>
> * it is declared immediately within the same declaration list as D or any partial or incomplete view of D; and
>
> * at least one of its operands is an access parameter with designated type D.

I'm not at all clear why the example code is legal, or why it would be legal to call it; since 'access Cell' appears to match "neither operand shall be of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator ..."

Might explain why compiling this example with GNAT (CE 2018) results in stack overflow.

From: Simon Wright <simon@pushface.org>
Subject: Re: Overloading operator “=” for anonymous access types?
Newsgroups: comp.lang.ada
Date: Sat, 12 Jan 2019 14:01:43 +0000

> I'm not at all clear why the example code is legal, or why it would be legal to call it; since 'access Cell' appears to match "neither operand shall be of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator ..."

Still not clear.

Note to self: do *not* attempt to define "=" for anonymous access types!

Would have liked the AIs to have said "it is illegal to define "=" for anonymous access types".

From: daicrkk@googlemail.com
Subject: Re: Overloading operator “=” for anonymous access types?
Newsgroups: comp.lang.ada
Date: Sat, 12 Jan 2019 07:15:38 -0800

> [...]

> I'm not at all clear why the example code is legal, or why it would be legal to call it; since 'access Cell' appears to match "neither operand shall be of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator ..."

I second that. Access Cell is an access-to-object type whose designated type is Cell (check), Cell has a user-defined primitive equality operator (check) such that its result type is Boolean (check), it is declared immediately within the same declaration list as Cell (check), at least one of its operands is an access parameter with designated type Cell (both operands are, check).

According to 4.5.2, universal_access "=" should never be allowed to kick in at all here, not even with "L = null". Or am I missing something?

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Overloading operator "=" for anonymous access types?
Newsgroups: comp.lang.ada
Date: Mon, 14 Jan 2019 17:08:32 -0600

>I second that. Access Cell is an access-to-object type whose designated type is Cell (check), Cell has a user-defined primitive equality operator (check) such that its result type is Boolean (check), it is declared immediately within the same declaration list as Cell (check), at least one of its operands is an access parameter with designated type Cell (both operands are, check).

>According to 4.5.2, universal_access "=" should never be allowed to kick in at all here, not even with "L = null". Or am I missing something?

Yup, I agree with this. My first thought when reading that example is that it is wrong, because I don't remember anywhere in Ada where the same operator with arguments of the same type means different things. I don't think the use of "null" could change that.

Dunno if John wrote that for a different version of Ada, or he was just confused by a rule that barely makes sense anyway.

As always, best avoid anonymous access types unless you have to use one of their special features (dynamic accessibility, dispatching, special discriminant accessibility, or closures [for access-to-subprograms]). And better still, lets lobby to get those special features optionally available for named access types so no one ever has to use an anonymous anything. :-)

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: Overloading operator "=" for anonymous access types?
Newsgroups: comp.lang.ada
Date: Mon, 14 Jan 2019 16:34:42 -0800

> As always, best avoid anonymous access types unless you have to use one of their special features (dynamic accessibility, dispatching, special discriminant accessibility, or closures [for access-to-subprograms]). And better still, lets lobby to get those special features optionally available for named access types so no one ever has to use an anonymous anything. :-)

Well, I'm all for getting rid of anonymous access types altogether -- though that might not be acceptable to the rest of the ARG as it would make previously-valid Ada non-valid, I think reducing the complexity of the language (and reduce instances of "a rule that barely makes sense anyway").

I thought there was an AI for first class subprograms / subprogram types, but I couldn't find it with a quick search... so either I'm misremembering or I'm just hitting all the wrong keywords in the search.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: Overloading operator "=" for anonymous access types?
Newsgroups: comp.lang.ada
Date: Tue, 15 Jan 2019 09:38:11 +0100

> Yup, I agree with this. My first thought when reading that example is that it is wrong, because I don't remember anywhere in Ada where the same operator with arguments of the same type means different things. I don't think the use of "null" could change that.

But the types are not same. It is universal_access vs. access.

> Dunno if John wrote that for a different version of Ada, or he was just confused by a rule that barely makes sense anyway.

> As always, best avoid anonymous access types unless you have to use one of their special features (dynamic accessibility, dispatching, special discriminant accessibility, or closures [for access-to-subprograms]). And better still, lets lobby to get those special features optionally available for named access types so no one ever has to use an anonymous anything. :-)

Named or anonymous it makes little difference, IMO.

Here is a classic multi-method case. "=" is such an operation. null is universal_access (4.2). For any access type P there are 3 equality operations "=":

```
function "=" (Left, Right : universal_access)
return Boolean;
   type P is access T;
   function "=" (Left : P; Right :
universal_access) return Boolean;
   function "=" (Left : universal_access; Right
: P) return Boolean;
   function "=" (Left, Right : P) return
Boolean;
```

When the last one is overridden, what happens with the second and the third?

One of three possibilities:

1. It inherits the base operation:

```
   function "=" (Left : P; Right :
universal_access) return Boolean is
   begin
      return universal_access (Left) = Right;
   end "=";
```

2. It silently overrides:

```
   function "=" (Left : P; Right :
universal_access) return Boolean is
   begin
      return Left = P (Right);
   end "=";
```

3. It gets overridden abstract and comparison to null becomes illegal because the operation is not defined.

[The reference manual is shy to say anything about it. It claims that universal_access is kind of class-wide, which would mean, if taken seriously, that "=" overloads and must clash with the original "=". Since it does not, universal_access is more like a parent type than class-wide.]

It seems that in the OP's case as in the case with named access types #2 is in effect, which is illogical, inconsistent, unsafe, but would be expected by most people.

Barnes' code presumes rather #1, which is logical, but confusing and error-prone.

#3 would be consistent and safe:

```
   if Ptr_Value = Ptr_Type (null) then --
Type conversion required
```

But it would not work with anonymous access types. So, if #3 were adopted, then overriding for anonymous types must be banished.

All this is fine and good, except that overriding

```
   function "=" (Left, Right : access T)
return Boolean;
```

is also a primitive of T! You cannot banish it.

P.S. And, wouldn't it be better to fix the type system, no?

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Overloading operator "=" for anonymous access types?
Newsgroups: comp.lang.ada
Date: Tue, 15 Jan 2019 15:00:31 -0600

> [The reference manual is shy to say anything about it. It claims that universal_access is kind of class-wide, which would mean, if taken seriously, that "=" overloads and must clash with the original "=".

This is what happens. However, such a clash would mean that you could never write a user-defined "=" for an anonymous access type. That would have been a good idea, but it would have to have been enforced with a Legality Rule to be sensible. Some thought that bad because of compatibility, so...

> Since it does not, universal_access is more like a parent type than class-wide.]

...there is a hack to have a preference for the user-defined one. That doesn't change the fact that universal_access is class-wide, it just make it possible to write a user-defined operator.

>P.S. And, wouldn't it be better to fix the type system, no?

This wart would be one of the things that would make "fixing the type system" so much harder. A proper solution (and the one we should have used in the first place) is to declare a "=" for every access type. I think we wanted to avoid that as anonymous access can be declared in places where declarations aren't allowed, so we came up with something worse. :-)

It's the idea of anonymous access types that destroys the type system that you have in mind. Your system keeps the types and operations together, and that makes no sense for an anonymous type (what are the operations for an anonymous type, and where are they declared? Any answer is either magical or nonsense.)

One has to get rid of nonsense things before one could regularize the type system, especially upon the lines you have been suggesting for years. It's not really possible for Ada; you would end up with an Ada-like language.

This is just another Ada

Return types

From: danielcheagle@gmail.com
Subject: ? Is ok return a type derived from ada.finalization.controlled from a "Pure_Function" ? thanks.
Newsgroups: comp.lang.ada
Date: Thu, 24 Jan 2019 15:56:10 -0800

Is ok return a type derived from ada.finalization.controlled from a function declared "Pure_Function" ?

Or yet, is ok declare a fuction returning a controlled type as "pure_function" ?

Thanks in Advance!!!

note1 : the type has a access value.

note2 : initialize, adjust and finalize overrided and working :-)

fragment example code:

```
--------------------------------
pragma Ada_2012;
pragma Detect_Blocking;

with Ada.Finalization;

package Arbitrary
  with preelaborate
is

   type Arbitrary_Type (size : Positive) is
     new Ada.Finalization.Controlled with
private;

   function To_Arbitrary (value : Integer;
             precision : Integer)
     return Arbitrary_Type
      with inline; -- Can I add "pure_function" ?

private

  type Mantissa_Type is array (Positive
             range <>) of Integer;
  type Mantissa_Pointer is access
             Mantissa_Type;

  type Arbitrary_Type (size : Positive) is
    new Ada.Finalization.Controlled with
record
     mantissa    : Mantissa_Pointer;
     exponent    : Integer;
     sign        : Integer range -1 .. 1;
     precision   : Positive := size;
   end record;

end arbitrary;

-----------------------------------------------

pragma Ada_2012;
pragma Detect_Blocking;

with Ada.Unchecked_Deallocation;

package body Arbitrary is

  procedure Delete is new
             Ada.Unchecked_Deallocation
             (Mantissa_Type,
               Mantissa_Pointer);
 ---------------------------------------------------------
  -- Initialize an Arbitrary_Type
  ---------------------------------------------------------
  procedure Initialize (Object : in out
             Arbitrary_Type) is
  begin
    Object.mantissa := new Mantissa_Type
             (1 .. Object.precision);
    Object.exponent     := 0;
    Object.sign         := 1;
    -- "here" for diminish race condition from
    -- OS' s
    Object.mantissa.all := (others => 0);
  end Initialize;

  ---------------------------------------------------------
  -- Fix an Arbitrary_Type after being  --
  -- assigned a value
  ---------------------------------------------------------
  procedure Adjust (Object : in out
             Arbitrary_Type) is
  begin
    Object.mantissa := new
          Mantissa_Type'(Object.mantissa.all);
  end Adjust;

 ---------------------------------------------------------
  -- Release an Arbitrary_Type;
 ---------------------------------------------------------
  procedure Finalize (Object : in out
             Arbitrary_Type) is
  begin
    if Object.mantissa /= null then
      Delete (Object.mantissa);
    end if;
    Object.mantissa := null;
  end Finalize;

 ---------------------------------------------------------
  -- Convert an Integer type to an
  -- Arbitrary_Type
 ---------------------------------------------------------
  function To_Arbitrary (value : Integer;
             precision : Integer)
    return Arbitrary_Type is
    result    : Arbitrary_Type (precision);
  begin
    result.mantissa (result.exponent + 1) :=
             value;
    Normalize (result);
    return result;
  end To_Arbitrary;

end arbitrary;
```

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: ? Is ok return a type derived from ada.finalization.controlled from a "Pure_Function" ? thanks.
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 15:20:47 -0600

Of course it's OK, "Pure_Function" is some GNAT-specific nonsense. :-)

My recollection is that GNAT does not check if Pure_Function makes sense, so the only question is whether you can live with the possible implications. (And I don't know why you would want to use Pure_Function anyway.)

Note that in Ada 2020, you would use the Global aspect to declare the usage of globals by your subprogram, and those are checked, so either the aspect is legal or your program won't compile. But GNAT hasn't implemented that yet, so far as I know.

From: Shark8 <onewingedshark@gmail.com>
Subject: Re: ? Is ok return a type derived from ada.finalization.controlled from a "Pure_Function" ? thanks.
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 16:22:33 -0800

IIRC, Pure_Function doesn't need to be in a Pure unit to be tagged as such, and the GNAT-specific meaning is: given a call with a particular set of parameter-values always returns the same result.

As I recall GNAT doesn't actually check this is case, but rather uses it for optimization purposes.

> Or yet, is ok declare a function returning a controlled type as "pure_function" ?

See above: "Pure_Function" has nothing to do with categorization or restrictions and is just an attribute denoting allowance for certain optimizations. (Again, IIRC.)

From: Simon Wright <simon@pushface.org>
Subject: Re: ? Is ok return a type derived from ada.finalization.controlled from a "Pure_Function" ? thanks.
Newsgroups: comp.lang.ada
Date: Sat, 26 Jan 2019 11:48:46 +0000

Given that the documentation of Pure_Function[1] says

... the compiler can assume that there are no side effects, and in particular that two calls with identical arguments produce the same result

and that

... there are no static checks to try to ensure that this promise is met

it would be a Bad Idea to apply it to your function.

[1] https://gcc.gnu.org/onlinedocs/gnat_rm/Pragma-Pure_005fFunction.html

Forbid local generic instantiations

From: joakimds@kth.se
Subject: Why forbid local generic instantiations?
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 01:43:29 -0800

[...]

Consider the following code:

```
procedure Main is
   package Integer_Vectors is new
Ada.Containers.Vectors (Positive, Integer);
begin
   null;
end Main;
```

It has a generic package instantiation local to the subprogram Main and not defined on package level. Both in AdaControl and GNATCheck there are rules to forbid local generic instantiations.

For example GNATCheck:

23.7.25 Generics_In_Subprograms

Flag each declaration of a generic unit in a subprogram. Generic declarations in the bodies of generic subprograms are also flagged. A generic unit nested in another generic unit is not flagged. If a generic unit is declared in a local package that is declared in a subprogram body, the generic unit is flagged.

This rule has no parameters.

Using AdaControl one can use the following rule to detect instantiations of generic packages/subprograms:

5.10 Declarations

This rule controls usage of various kinds of declarations, possibly only those occurring at specified locations.

...

Why is it considered bad practise to use local generic instantiations? Within the C++ Community, limiting the use of templates doesn't seem an issue. On the contrary, going all in with template metaprogramming is the norm.

Does local generic instantiations have a performance penalty? Is it something that may be error-prone? Limit cross-compiler compatibility? Why does the rule exist to ban local instantiations? I've been googling/searching the web for an answer to this question but have not found an explanation. Does anybody know?

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Subject: Re: Why forbid local generic instantiations?
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 17:36:33 +0100

> Why is it considered bad practise to use local generic instantiations? Within the C++ Community, limiting the use of templates doesn't seem an issue. On the contrary, going all in with template metaprogramming is the norm.

It isn't bad practice. Mostly such rules are premature optimization. Are there rules against regular pkgs in such places?

There's no difference.

It makes perfect sense for things to be declared in the smallest scope in which they're needed. This is true of anything, not just pkgs.

A pkg in a subprogram is elaborated every time the subprogram is called. If the elaboration of a specific pkg is expensive and timing requirements are tight, it might make sense to move that pkg to a larger scope. But a general rule against them for "efficiency" doesn't make sense. Limiting it to pkgs that are generic instantiations makes less sense.

Perhaps such people don't know that instantiation takes place during compilation and has no run-time impact.

As a 1st-order approximation, anything the "C++ Community" does should be avoided.

From: "Randy Brukardt" <randy@rrsoftware.com>
Subject: Re: Why forbid local generic instantiations?
Newsgroups: comp.lang.ada
Date: Fri, 25 Jan 2019 15:23:55 -0600

> Perhaps such people don't know that instantiation takes place during compilation and has no run-time impact.

I agree with most of what you said, but this statement is false, since the instance is elaborated at the point of the instantiation. Depending on the generic, that could be a substantial amount of execution time. (Note that is even more true for a code-shared implementation like Janus/Ada, since the elaboration of the instance creates the instantiation descriptor.)

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Subject: Re: Why forbid local generic instantiations?
Newsgroups: comp.lang.ada
Date: Sat, 26 Jan 2019 10:56:27 +0100

> [...]

I can't tell from what you've written if what I said is wrong or if we're saying basically the same thing in different ways. I'm not familiar with the way shared-code generics are instantiated. Macro-expansion instantiation is straightforward.

The rule I learned (Ada 83) was: Instantiation happens during compilation; elaboration happens during run time.

In more detail: Instantiation is the process whereby a compiler effectively replaces an instantiation with a regular pkg (the instance). The result is no different from having written the resulting regular pkg instead of the instantiation, except for possible code sharing with other instantiations of the same generic

[ignoring the case of an instantiation in a pkg spec].

All pkgs, regular or generic instances, are elaborated during run time. That elaboration can be as complex as the developer wants. In the case of a pkg in a subprogram, that elaboration happens every time the subprogram is called.

That's what I learned back when dinosaurs ruled the earth. I gather from what you've written that a shared-code compiler may increase the amount of elaboration by some (hopefully small, fixed?) amount, so it's not technically correct unless the increase is small enough to be considered negligible. I think it's correct for compilers that do macro-expansion instantiation, and close enough for the rule to be correct as a 1st-order approximation.

If I'm wrong, I'd like to be corrected.

From: Jere <jhb.chat@gmail.com>
Subject: Private extension of a synchronized interface
Newsgroups: comp.lang.ada
Date: Fri, 15 Feb 2019 16:52:07 -0800

I'll get to my ultimate goal later, but while following various rabbit trails, I came across a situation I couldn't solve. GNAT allows you to make private extensions to synchronized interfaces and it allows you to complete those private extensions with protected types. I can't, however, figure out how it overrides the abstract procedures and functions of the synchronized interface.

If I don't specify an override and try to call the procedure, it complains that the procedure is abstract. If I try to override the abstract function, it complains that the signature doesn't match the one in the protected body. I don't know if this is a GNAT issue or something that Ada doesn't allow. Here is some test code. It compiles as is, but there are two parts that if you uncomment either one of those it fails to compile.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
   package Example is

      type An_Interface is synchronized
            interface;
      procedure p1(Self : in out
            An_Interface)  is abstract;

      type Instance is synchronized new
            An_Interface with private;

      -- The following lines give the errors:
      -- "p1" conflicts with declaration at line
      --- xxx and missing body for "p1"

      --overriding
      --procedure p1(Self : in out Instance);

   private
      -- Some hidden implementation types,
      -- constants, etc.

      -- Instance full view is a protected type
      protected type Instance is new
            An_Interface with
         procedure p1;
      private
         -- some hidden stuff;
      end Instance;

   end Example;

   package body Example is
         protected body Instance is
         procedure p1 is
         begin
            Put_Line("Did Something");
         end p1;
      end Instance;

   end Example;

   v : Example.Instance;

begin
  Put_Line("Hello, world!");
  -- The following line gives the error:
  -- call to abstract procedure must be
  -- dispatching
    --v.p1;
end Hello;
```

My ultimate goal is not having to declare a bunch of extra types and packages in the

public view to only use them in the private view of the protected object. I'd prefer that all of the private stuff actually be in a private section. So I'm not tied to interfaces, but it was one attempt at getting stuff moved down to the private section. But while I went down the interfaces rabbit hole, I just found the issue I ran into odd.

Does anyone know how to create the correct overrides for the example above?

Extension of synchronized interfaces

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: Private extension of a synchronized interface
Newsgroups: comp.lang.ada
Date: Sun, 17 Feb 2019 10:50:21 +0100

> I'll get to my ultimate goal later, but while following various rabbit trails, I came across a situation I couldn't solve. GNAT allows you to make private extensions to synchronized interfaces and it allows you to complete those private extensions with protected types. I can't, however, figure out how it overrides the abstract procedures and functions of the synchronized interface.

> If I don't specify an override and try to call the procedure, it complains that the procedure is abstract. If I try to override the abstract function, it complains that the signature doesn't match the one in the protected body. I don't know if this is a GNAT issue or something that Ada doesn't allow. Here is some test code. It compiles as is, but there are two parts that if you uncomment either one of those it fails to compile.

Reading RM 9.5.2 (13.2/2) does not really help:

"if the overriding_indicator is overriding, then the entry shall implement an inherited subprogram;"

An inherited subprogram is already implemented per, well, inheritance. May be it means:

1. shall implement a primitive operation (it overrides here);

2. shall implement an overridden primitive operation (it implements overriding declared earlier).

Neither #1 nor #2 work.

But synchronized interfaces are totally bogus from the software design POV. It is a pure implementation aspect exposed. Why do you care?

Aggregate a protected object and delegate primitive operations to it.

From: Jere <jhb.chat@gmail.com>
Subject: Re: Private extension of a synchronized interface
Newsgroups: comp.lang.ada
Date: Sun, 17 Feb 2019 05:46:17 -0800

> But synchronized interfaces are totally bogus from the software design POV. It is a pure implementation aspect exposed. Why do you care?

> Aggregate a protected object and delegate primitive operations to it.

That's what I am doing as my own solution. I was intrigued with the code above as an alternate solution because it could potentially give a compile time indication that a procedure was a protected operation (as opposed to me relying on simply providing that via comments). A delegate non protected procedure has to rely on the comment. I didn't even want the interface to use as an interface, just as a means to at the API level to have a compiler enforced indication that the procedure was from a protected object. I started with a protected object in the public view but the implementation details of the private part of the protected object led to about 10 lines of code (type declarations and a couple of package specifications) that had no use to the public view but had to be there because of how protected object declarations work. I saw this as a potential means of information hiding. My actual solution is as you suggested with delegate operations that call the protected object. However, I honestly wanted to know why Ada allowed one to setup the private extension but not allow you to actually provide the functions (or if this was a GNAT issue or if I was just not using the right syntax). So the reason I care was a thirst for knowledge of how things work.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: Re: Private extension of a synchronized interface
Newsgroups: comp.lang.ada
Date: Sun, 17 Feb 2019 15:52:38 +0100

Given to who? The compiler knows already, the user should not care. It is an implementation aspect which simply does not belong here.

What could make sense is an entry interface, a primitive operation which could be queued/requeued to, used in timed entry call etc.

> A delegate non protected procedure has to rely on the comment.

There is no contract that could require it protected. It is a property of the object/task and no property of an operation. You could not do anything with a task or protected object that would not resolve into a protected action anyway.

[...]

> However, I honestly wanted to know why Ada allowed one to setup the private extension but not allow you to actually provide the functions (or if this was a GNAT issue or if I was just not using the right syntax). So the reason I care was a thirst for knowledge of how things work.

Ada 2005 stuff, most of it makes little sense to me. It was some halfhearted attempt to unite tagged types with tasks and protected objects with no desire to actually do that...

From: Jere <jhb.chat@gmail.com>
Subject: Re: Private extension of a synchronized interface
Newsgroups: comp.lang.ada
Date: Sun, 17 Feb 2019 07:36:18 -0800

The compiler cannot always tell depending on how and where you call buried protected operations. I always prefer compile time catching over run time catching.

> > A delegate non protected procedure has to rely on the comment.

> There is no contract that could require it protected. It is a property of the object/task and no property of an operation. You could not do anything with a task or protected object that would not resolve into a protected action anyway.

Protected procedures/functions/entries are particularly heavy operations.

I don't know if you generally work in low level embedded environments, but being able know and plan for that can be very critical. It can change how you approach your design. When you work in systems where your system clock is 1-4MHz, timing of operations does start to matter.

> > However, I honestly wanted to know why Ada allowed one to setup the private extension but not allow you to actually provide the functions (or if this was a GNAT issue or if I was just not using the right syntax). So the reason I care was a thirst for knowledge of how things work.

> Ada 2005 stuff, most of it makes little sense to me. It was some halfhearted attempt to unite tagged types with tasks and protected objects with no desire to actually do that...

I'm just curious if or why the process was stopped half way instead of abandoned or completed (again that is assuming I didn't use the wrong syntax, in which case it's simply that I'm structuring the syntax wrong).

I don't really need to marry them with tagged types. I do appreciate the ability to dispatch over a group of related but different tasks much more easily and the interfaces give that. The way that Ada chose to implement interfaces is one of many ways (not all of which would have required tagged types).