## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editorial Policy for <em>Ada User Journal</em></td>
<td>2</td>
</tr>
<tr>
<td>Editorial</td>
<td>3</td>
</tr>
<tr>
<td>Quarterly News Digest</td>
<td>5</td>
</tr>
<tr>
<td>Conference Calendar</td>
<td>22</td>
</tr>
<tr>
<td>Forthcoming Events</td>
<td>29</td>
</tr>
<tr>
<td>Press Release</td>
<td></td>
</tr>
<tr>
<td>&quot;<em>Ada 2012 Language Standard Approved by ISO</em>&quot;</td>
<td>35</td>
</tr>
<tr>
<td>Special Contribution</td>
<td></td>
</tr>
<tr>
<td>J. G. P. Barnes</td>
<td></td>
</tr>
<tr>
<td>“Rationale for Ada 2012: 6 Predefined Library”</td>
<td>38</td>
</tr>
<tr>
<td>Ada-Europe 2012 Tutorials</td>
<td></td>
</tr>
<tr>
<td>J.-P. Rosen</td>
<td></td>
</tr>
<tr>
<td>“Designing and Checking Coding Standards for Ada”</td>
<td>45</td>
</tr>
<tr>
<td>M. Aldea Rivas</td>
<td></td>
</tr>
<tr>
<td>“Advanced Ada Support for Real-Time Programming”</td>
<td>49</td>
</tr>
<tr>
<td>Ada Gems</td>
<td>57</td>
</tr>
<tr>
<td>Ada-Europe Associate Members (National Ada Organizations)</td>
<td>60</td>
</tr>
<tr>
<td>Ada-Europe 2012 Sponsors</td>
<td></td>
</tr>
</tbody>
</table>
Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

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

Proceedings and Special Issues

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

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

Reviews

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

News and Product Announcements

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

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.

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

Commentaries

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

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

Announcements and Reports

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

Submission Guidelines

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

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.
In this first Editorial of 2013, I would like to start by drawing your attention to the preliminary program of the Ada-Europe conference, which will take place June 10-14, in Berlin, Germany. You can find this preliminary program in the forthcoming events section of this issue; details will be increasingly available in the conference website.

The conference program includes three very valuable keynotes (on Tuesday, Bruce Douglass, Chief Evangelist of IBM Rational with a keynote about “Model-based Ada Development for DO-178B/C and the Application of Agile Methods”; on Wednesday, Jack Ganssle, of The Ganssle Group, on “The Way Ahead in Software Engineering: Replacing Artists With Disciplined Grownups”; and Thursday, Giorgio Buttazzo, from the Scuola Superiore Sant’Anna of Pisa, Italy, on “Research Challenges in Exploiting Multi-Core Platforms for Real-Time Applications”), a panel concerning the use of Heap in Real-Time Systems, six sessions of technical papers and industrial presentations, a session presenting the main changes and improvements with Ada 2012, and an extensive group of tutorials on Monday and Friday. The week will also have an interesting social program, in particular the Wednesday dinner at the Botanical Garden. I am looking forward to meet you all in Berlin!

And this year is full of Ada-focused events. After the Ada Developer Room at FOSDEM, last February, and that we mentioned in the last issue, we will have a new edition of the International Real-Time Workshop, which will take place in the wonderful York, UK, April 17-19. One week after, in April 25th, the Ada Conference UK 2013 takes place in Birmingham, UK, and finally the ACM SIGAda High Integrity Language Technology conference, taking place fall 2013, in Pittsburgh, USA. Information in the latter two can also be found in the forthcoming events section.

As for the technical content of this issue, we continue the publication of the Ada 2012 Rationale, a very valuable contribution to the Ada community by John Barnes. Chapter 6 describes some improvements done to the predefined library, except for containers; changes there will be presented in the next issue.

The issue continues with contributions derived from the Ada-Europe 2012 conference: two articles presenting summaries from tutorials. In the first, Jean-Pierre Rosen, from Adalog, France, presents an overview of the problems encountered with coding standards, and gives some experience hints on how to have rules that are understood, and also accepted and used. In the second article, Mario Aldea Rivas, from Universidad de Cantabria, Spain, presents an overview of the mechanisms for real-time programming in Ada, focusing on the new approaches provided by Ada 2005 and Ada 2012.

We continue with contributions from the Gem of the Week; in this issue the series of gems from Bob Duff, of AdaCore, which clarify what is the notion of “erroneous” in Ada. And, as usual, the reader will also encounter the information provided in the News Digest and Calendar sections.

Luis Miguel Pinho
Porto
March 2013
Email: AUJ_Editor@Ada-Europe.org
Quarterly News Digest

Jacob Sparre Andersen
Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents
Ada-related Organizations 5
Ada-related Events 5
Ada Semantic Interface Specification 5
Ada-related Resources 6
Ada-related Tools 6
Ada-related Products 12
Ada and GNU/Linux 13
Ada and Microsoft 13
References to Publications 14
Ada Inside 16
Ada in Context 17

Ada-related Organizations

Ada 2012 Standard approved by ISO

From: Dirk Craeynest
<dirk@vana.cs.kuleuven.be>
Date: Tue, 18 Dec 2012 07:45:44
Subject: Press Release - Ada 2012
Language Standard Approved by ISO
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc
FOR IMMEDIATE RELEASE
Ada 2012 Language Standard Approved by ISO
Language revision adds contract-based programming, multicore support, and other advanced features
GENEVA, Switzerland, December 18, 2012 - The Ada Resource Association (ARA) and Ada-Europe today announced the approval and publication of the latest version of the Ada programming language by the Geneva-based International Organization for Standardization (ISO). The language revision, known as Ada 2012, was under the auspices of ISO/IEC JTC1/SC22/WG9 and was conducted by the Ada Rapporteur Group (ARG) subunit of WG9, with sponsorship in part from the ARA and Ada-Europe. The formal approval of the standard was issued on November 20 by ISO/IEC JTC 1, and the standard was published on December 15. 
[see also the full press release included in this issue. —sparre]

Ada-related Events

Course at Open Source Days in Copenhagen

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon, 13 Jan 2013
Subject: Parallel programming for sequential programmers.
Assuming you already know Sequential programming in Ada. -- Just enough to allow you to map your existing programming knowledge to Ada. -- I.e. parallel programming without interaction.
- Communication between tasks using safe, shared objects. One kind of interaction between tasks.
- Direct communication between tasks using rendezvous. -- Another kind of interaction between tasks.
- Patterns for implementing parallel execution.

FOSDEM 2013

From: Dirk Craeynest
<dirk@cs.kuleuven.be>
Date: Wed, 13 Feb 2013
Subject: Ada Developer Room at FOSDEM
All presentations from the Ada Developer Room, held at FOSDEM 2013 in Brussels recently, are available on the Ada-Belgium web site now. A few pictures taken during the event are on-line as well.

- "Welcome & Ada-Europe info" by Dirk Craeynest - Ada-Belgium
- "Introduction to Ada for Beginning and Experienced Programmers" by Jean-Pierre Rosen - AdaCore.
- "Ada on Android" by José F. Ruiz - AdaCore.
- "Ada Tasking: Multithreading Made Easy" by Ludovic Brenta - Debian.
- "Compile-Time Dimensionality Checking" by José F. Ruiz - AdaCore.
- "Telephone Reception Management with Alice on Pi?" by Kim Rostgaard Christensen - AdaHeads K/S.
- "Simplifying the Use of Formal Methods" by Valentine Reboul - AdaCore.

Presentation abstracts, copies of slides, speakers bios, pointers to relevant information, links to other sites, some pictures, etc., are all available on the Ada-Belgium site[1].

130203-fosdem.html>

Ada Semantic Interface Specification (ASIS)

ASIS 2012

From: Peter C. Chapin
<PChapin@vtc.vsc.edu>
Date: Thu, 13 Dec 2012 19:01:39 -0500
Subject: ASIS 2012
Newsgroups: comp.lang.ada

I'm wondering if there is any work being done on an update of the ASIS standard to support Ada 2012.

From: Jean-Pierre Rosen
<rosen@adalog.fr>
Date: Fri, 14 Dec 2012 06:24:50 +0100
Subject: Re: ASIS 2012
Newsgroups: comp.lang.ada

Yes. Now that Ada 2012 is out, the ARG is working on ASIS. The decision was taken to skip Ada2005, and go directly for 2012.
Ada-related Resources

Smart Pointers
From: Christoph Karl Walter Grein
<christ-usch.grein@t-online.de>
Date: Wed, 10 Oct 2012
Subject: Smart Pointers
URL: http://www.christ-usch-grein.
        homepage.t-online.de/Ada/
        Smart_Pointers.html

Access types in Ada have been designed in a way to prevent the occurrence of dangling references, i.e. they can never designate objects that have gone out of scope. There remains however one problem: When Unchecked_Deallocation is used in order to reclaim storage, we might access already freed storage or, even worse, storage occupied by new objects of different types, unless utmost care is taken (this is the very reason why the generic is called "unchecked" deallocation).

[...] [With source download. —sparre]

How tall is a kilogram
From: Vincent Pucci
Date: Mon, 12 Nov 2012
Subject: Gem #136: How tall is a kilogram?
URL: http://libre.adacore.com/
        adaanswers/gems_single/
        gem-136-how-tall-is-a-kilogram

This Gem outlines the new GNAT dimensionality checking system. This feature relies on Ada 2012 aspect specifications, and is available from version 7.0.1 of GNAT onwards. The GNAT compiler now supports dimensionality checking. Ever since the appearance of user-defined operators in Ada 83 there have been attempts to use these to declare types with dimensions, and perform dimensionality checks on scientific code. These attempts were unfortunately unwieldy and were not adopted by the language. The system we describe here is lightweight and relies on the aspect specifications introduced in Ada 2012. We hope that the engineering and scientific community will find it convenient and easy to use.

[...] [A 3.8 file.]

Ada Sub-Reddit has more than 500 members
From: Marc Criley
<mc.provisional@gmail.com>
Date: Mon, 24 Dec 2012
Subject: Ada Sub-Reddit Barrels through the 500 Members Threshold?
URL: http://www.reddit.com/r/Ada

Thanks to a pair of postings in Reddit's general programming forum regarding the official release of the Ada 2012 standard that drew attention and a lot of upvotes -- and my plug in the comments section for the Ada sub-reddit [1], r/ada added enough members to now exceed 500.

It's a nice milestone to have hit, and serves to confirm that interest in the language continues to grow.


Ada 2012 Reference Manual in info format
From: Stephen Leake
<stephen_leake@stephe-leake.org>
Date: Wed, 26 Dec 2012 15:44:30 -0500
Subject: final version of Ada Reference Manual in info format
Newsgroups: comp.lang.ada

The final version of the Ada Reference Manual in info format is now available at http://stephe-leake.org/ada/arm.html

Counting seconds with attiny4313
From: Tero’s Arduino Blog
Date: Mon, 21 Jan 2013
Subject: Counting seconds with attiny4313
URL: http://arduino.adacore.com/
        counting-seconds-with-attiny4313.html

AVR-Ada includes AVR.Real_Time package and separate AVR.Real_Time.Clock function, whose job is to return the current time. On Arduino, the current time means seconds from the device bootup, in other words it tells you how long the device has been on.

[...] [Tero gives detailed instructions on how to show the running time of an Arduino on a 4x7 segment display. —sparre]

Ada community on Google+
From: Marc Hanisch and Thomas Locke
Date: Tue, 22 Jan 2013
Subject: Ada group on Google+ with 100+ members
URL: https://plus.google.com/
[The Ada community on Google+ has passed 100 members. —sparre]

Rosetta Code
From: Jacob Sparre Andersen
<jacob@jacob-sparre.dk>
Date: Thu, 24 Jan 2013
Subject: Ada on Rosetta Code
URL: https://bitbucket.org/
        ada_on_rosetta_code/solutions

Ada solutions to the tasks presented on Rosetta Code. All solutions included here build and run (and should ideally also include a test example).

[Not complete yet. Contributions are welcome. —sparre]

Arduino temperature measurement with DS18B20 and one-wire protocol
From: Tero’s Arduino Blog
Date: Tue, 05 Feb 2013
Subject: Temperature measurement with DS18B20 and one-wire protocol
URL: http://arduino.adacore.com/
        temperature-measurement-with-
        ds18b20-and-one-wire-protocol.html

Required parts:
- Arduino UNO or Duemilanove
- DS18B20 temperature sensor
- Some jumper wires
- About 5K ohm resistor (4.7Kohm or 4.99Kohm is ok)

I had a couple of extra DS18B20 temperature sensors lying around, so I was interested in communicating with them using Arduino. DS18B20 uses 1-wire protocol for communication and AVR-Ada supports it (1-wire and DS18B20) out of the box.

[...] [Tero gives detailed instructions on how to use an Arduino for making temperature measurements. —sparre]

Introduction to AWS
From: Thomas Locke <tl@ada-dk.org>
Date: Sun, 10 Feb 2013
Subject: Using the Ada Web Server (AWS), part 1
URL: http://blogs.fsf.de/thomaslocke/2013/
        02/10/using-the-ada-web-server-aws-
        part-1/

The Ada Web Server (AWS) is a big library with a whole lot of functionality, far more than I can manage to write about, so what I am going to do in this first article is focus on the basic stuff: Getting a server up and registering a couple of content handlers.

[...] [Thomas Locke gives a good introduction for developers who want to get started using AWS for implementing web servers. —sparre]

Ada-related Tools

Code sketcher for GtkAda 2.24
From: François Fabien
<francois.fabien@hotmail.com>
Date: Fri, 23 Nov 2012 14:28:39 +0100
Subject: [ANN] A code sketcher for GtkAda 2.24
To: gtkada@lists.adacore.com

Gate3[1] is an Ada code sketcher that generates an Ada prototype from a glade 3.8 file.
It extracts the signal handlers from the XML file and outputs a main procedure and callbacks packages.

Comes with a tutorial and 4 code samples.

A starter for newcomers to glade3.


**Turbo Pascal 7 emulation**

From: Pascal <p@p14@orange.fr>

Date: Sun, 2 Dec 2012 19:47:04 +0100

Subject: [gtkada] [ANN] TP7 emulation V2.7 with GtkAda

To: GtkAda mailing list

<gtkada@lists.adacore.com>

Hello, here are changes of TP7-Ada since last announcement in June.

Among changes:

- you can now use ESC key, left, right, home and end arrows with input line in text window,
- MouseNewPosition has been implemented
- graphics fonts from TURBO CHR are now translated in Ada (see screen captures)

TP7-Ada is a port of Turbo Pascal libraries in Ada with GtkAda support. Moreover it can be used as a basic multi-purpose library for simple text or graphic stuff with GtkAda.

See screen captures on:

http://blady.pagesperso-orange.fr/tp7/ada.html

The complete code is here:

http://p2ada.svn.sourceforge.net/viewvc/p2ada/extras/tp7ada/current/

All TP7 features are not completely functional, see current status:


All Pascal source codes were translated in Ada with P2Ada translator:

http://sourceforge.net/projects/p2ada/

Feel free to send any feedback.

**Generic Command Line Parser**

From: Riccardo Bernardini <framefritti@gmail.com>

Date: Fri, 14 Dec 2012 13:17:17 -0800

Subject: GCLP (Generic Command Line Parser) 1.0.0 released

Newsgroups: comp.lang.ada

I just released on Launchpad (https://launchpad.net/gclp/1.0/1.0.0) the very first release of GCLP (Generic Command Line Parser).

I almost can hear you: "Not yet another command line parser!" :-) Yes, I know that there are few command line parsing libraries, but I was not able to find one that I liked, so… I scratched my own itch.

A peculiarity of GCLP, that I find very convenient for commands with many parameters, is that the parameters are nominal and not positional. For example:

```
fictional.exe input=foo.txt output=bar.xml
```

From: Yannick Duchêne <yannick.duchene@yahoo.fr>

Date: Sat, 15 Dec 2012 00:33:41 +0100

Subject: Re: GCLP (Generic Command Line Parser) 1.0.0 released

Newsgroups: comp.lang.ada

If that's a real life example, then it does not follow the standard on Windows platform (as that's an *.exe), which is to use a slash prefix for parameters.

One nice feature of a command line parser I believe, is to allow to be close to the platform standard. There's already too much inconsistencies with too many applications not following any common standard. In return, this make the command line interaction more intuitive and straight away.

From: Riccardo Bernardini <framefritti@gmail.com>

Date: Sat, 15 Dec 2012 08:20:19 -0800

Subject: Re: GCLP (Generic Command Line Parser) 1.0.0 released

Newsgroups: comp.lang.ada

No, it is not a real life example. I just added .exe to emphasise that "fictional" was a command.

Do you mean something like:

```
fictional.exe /input=foo.txt /output=bar.xml
```

Well, you could do this as well, by setting the parameter names to "\"input\" and "\"output\". By the way, the real "translation" of this under Linux would be:

```
fictional.exe --input=foo.txt
--output=bar.xml
```

and you can do this as well, by using "--input" and "--output" as parameter names.

I agree that this differentiates itself from the usual syntax. I choose this approach (and wrote this package) once that I needed to write a program with a fairly complex syntax and with many possible variations. So, I decided for this approach that, you could say, have no (positional) parameters, but only options. In this case, the "-", "--" or "\"?\" necessary to mark an option became redundant. Nothing prevents you to put that back in the option name, if you desire.

From: Riccardo Bernardini <framefritti@gmail.com>

Date: Sat, 15 Dec 2012 10:08:05 -0800

Subject: Re: GCLP (Generic Command Line Parser) 1.0.0 released

Newsgroups: comp.lang.ada

If the command line become too complex, that may suggest the command is not well suited for the matter. May be a configuration file or project file, using XML, could be an option. […]

Yes, I understand, but between a simple command like:

```
cp src dst
```

and a command that makes it convenient to give parameters via a configuration file (maybe in XML format), there is a grey area where a positional syntax is too complex (was the port number the first parameter? or the third?), but the complexity of a configuration file is not justified (maybe also because you use the command only once). In that grey area a syntax like this, in my opinion, becomes convenient.

I like this syntax also because it makes calls in shell scripts a bit more readable. As a quasi-real life example, recently we wrote a program that takes a directory full of images, process them and write the result in another directory. A (more or less) realistic call for this program could be:

```
process input-dir=/foo/bar zoom=9 area=128x128 output-dir=/bar/foo
```

this is a bit more readable (inside a script) than:

```
process /foo/bar /bar/foo 9 128x128=20
```

Of course, if you prefer more standard names like "--input-dir", you can use them.

The example above shows what I mean with "complex syntax" (maybe it is not a perfect choice of words): a command that requires a fair number of parameters (say, more than 3-4), some of which can be optional. If the syntax was such that you need to express, say, conditionals or complex dependencies between parameters (the command line syntax of VLC comes to my mind :-/), then I agree that a structured configuration file is way better (but not in XML, please! :-) I hate to parse XML… :-)) even with specialised libraries.

From: Brian Drummond <brian@shapes.demon.co.uk>

Date: Tue, 18 Dec 2012 15:33:22 +0000

Subject: Re: GCLP (Generic Command Line Parser) 1.0.0 released

Newsgroups: comp.lang.ada

[…] Perhaps you can combine the two modes: the GCLP could read a complex command line, or some/all of the options - in the exact same syntax - from file. It would make refactoring complex command lines effortless…
Matreshka
From: Vadim Godunko
Date: Sat, 15 Dec 2012 07:50:46 -0800
Subject: Ann: Matreshka 0.4.0
Newsgroups: comp.lang.ada
We are pleased to announce new version of Matreshka framework. New version includes support for SOAP protocol and WS-Security extension, and WSDL to Ada translator. See ReleaseNotes for more information:
http://forge.ada-ru.org/matreshka/wiki/ReleaseNotes/0.4
Matreshka 0.4.0 can be downloaded as source tarball
http://forge.ada-ru.org/matreshka/wiki/Download
or as binary package for OpenSUSE, Fedora, and overlay for Gentoo.

Excel Writer
From: Gautier de Montmollin
Date: Mon, 17 Dec 2012 03:49:26 -0800
Subject: Ann: Excel Writer v.10
Newsgroups: comp.lang.ada
There is a new release of ExcelWriter[1]. What's new:
- Excel Writer is significantly faster, especially for numerical output.

Qt5Ada
From: Leonid Dulman
Date: Fri, 21 Dec 2012 08:30:10 -0800
Subject: Announce : Qt5Ada version 5.0.0
Newsgroups: comp.lang.ada
There is a new version of Zip-Ada on [http://unzip-ada.sf.net/].

Zip-Ada
From: Gautier de Montmollin
Date: Sat, 22 Dec 2012 12:17:09 -0800
Subject: Ann: Zip-Ada v.45
Newsgroups: comp.lang.ada
An update of the GWindows framework is packaged in an installer.
--- GWindows Setup 22-Dec-2012.exe @ http://sf.net/projects/gnavi/
The major addition is a contribution,
GWindows.Common Controls.Ex_List_View, with very cool features, by F. Maier.
Major changes in the framework - numbers below refer to svn repository revisions:
195: GWindows.Common_Dialogs: Get_Directory with optional initial path
180: GWindows.Windows: Drop Files support Unicode names
179: GWindows.Common_Dialogs: Open_File, Open_Files, Save_File, Get_Directory support Unicode names
173: GWindows.Common_Controls: List_View_Control_Type: added Column_Width function
170: GWindows.Common_Dialogs: added Open_Files
166: GWindows.Common_Controls: added method On_Item_Changed to List_View_Control_Type
158: GWindows.Common_Controls: List_View_Control_Type: Insert_Item also with Sorted_Index as 'out' parameter
157: GWindows framework compatible (again) with Ada 95

Ada Utility Library
From: Stephane Carrez
Date: Wed, 26 Dec 2012 22:36:17 +0100
Subject: [Ann] Ada Utility Library 1.6.0 is available
Newsgroups: comp.lang.ada
Ada Utility Library[1] is a collection of utility packages for Ada 2005. A new version is available which provides:
- Support for HTTP clients (curl, AWS, …)
- Support for REST APIs using JSON
- New operations To_JSON and From_JSON for easy object map serialisation
- Added a listeners to help implementing the observer/listener design patterns
- Added support for wildcard mapping in serialisation framework
- New option -d <dir> for the unit test harness to change the working directory,
- New example facebook.adb to show the REST support.
It has been compiled and ported on Linux, Windows and NetBSD (GCC 4.4, GNAT 2011, GCC 4.6.3).

Lapack
From: jpwodruff@gmail.com
Date: Thu, 27 Dec 2012 10:48:14 -0800
Subject: A thicker binding for Lapack
Newsgroups: comp.lang.ada
There is a complete thin binding for lapack available which was evidently started by Wasi Chaopanorn in about 1996 and was recently finished by Nasser Abbasi. There about 1000 subroutines; their parameter signatures exactly translate the Fortran, and each procedure is Pragma Imported.
I've been considering adding value to that binding to address two perceived problems:
The specification involving the integer index used for matrix types is inconsistent
between the binding and Ada.Numerics.Generic_Real_Arrays. The binding declares that type because the user's code must comply with Fortran's integer. The Ada compiler is obliged to use that same type. Unfortunately Generic_Real_Arrays uses the base type integer for index, so there is no expectation that the representation would be the same as Fortran-compiled library. Another issue to resolve is that lapack, being Fortran, defines column-major storage while Ada is row-major. Parameters passing between these two conventions evidently must be transposed. I think I have a technique for interoperating with the Ada.Numerics types. My plan involves "wrapping" the lapack routines in a layer that converts types and transposes matrices. Here is a typical procedure specification as translated directly from the Fortran, with my transformation to Ada standard types.

```
procedure SGESV
(N : Natural;
NRHS : Natural;
A : in out Real_Arrays.Real_Matrix;
-- instance of Generic_
LDA : Natural;
IPIV : out Integer_Vector;
B : in out Real_Arrays.Real_Matrix;
LDB : Natural;
INFO : out Integer);

The math requirements for this procedure are very much like the "Solve" that was discussed in a recent thread.

-- function Solve (A : Real_Matrix; X : Real_Vector) return Real_Vector;

The B parameter in sgesv resembles X in the "Why X as argument name?" discussion. I am trying to make the lapack item as serviceable as the gnat standard item.

My trials at maintaining parameter definitions that echo the Fortran binding have led only to grief. The integer inputs N, NRHS, LDA, LDB make precious little sense, considering the transposition of the matrices that takes place inside the body of procedure sgesv.

After several drafts I'm thinking that a thick binding might look like this:

```
procedure SGESV
(A : in out Real_Arrays.Real_Matrix;
IPIV : out Integer_Vector;
B : in out Real_Arrays.Real_Matrix;
INFO : out Integer);
```

Programming with Ada attributes seems to clarify the relations between the several parameters. Now I think that the integer parameters will be computed inside the body of the overlay procedure. Still working on the details.

Guided by the spiral model I'll try a couple examples. Later if there is a benefit to Ada programmers interested in numeric analysis I'll address the scale of the massive lapack library.

Suggestions are welcome!

From: Shark8
<oneingedshark@gmail.com>
Date: Thu, 27 Dec 2012 11:18:09 -0800
Subject: Re: A thicker binding for Lapack Newsgroups: comp.lang.ada

> [column-major vs. row-major]

This is a non-issue; Ada allows for the specification of row- or column-major ordering via the Convention pragma (or aspect).

Example:

```
B : array (Integer Range <>)
   of Integer;
   Integer Range <>)

IPIV : out Integer_Vector;
B : in out Real_Arrays.Real_Matrix;
LDB : Natural;
INFO : out Integer);
```

There may be some deep reason for making A in-out, but if you don't see a need to preserve the internal LU decomposition I'd make a copy internally, and also keep IPIV internal.

Also, I'd make B 'in' and use a third "out" parameter Result.

And, perhaps INFO should be replaced by Constraint_Error?

From: ipvoodruuff@gmail.com
Date: Sat, 29 Dec 2012 10:36:01 -0800
Subject: Re: A thicker binding for Lapack Newsgroups: comp.lang.ada

> procedure SGESV
(A : in out Real_Arrays.Real_Matrix;
IPIV : out Integer_Vector;
B : in out Real_Arrays.Real_Matrix;
INFO : out Integer);
```

The alignment is visible to the user through (at least) the program's I/O.

Conclusion: transpose appears necessary.

From: Simon Wright
<simon@pushface.org>
Date: Sat, 29 Dec 2012 10:59:46 +0000
Subject: Re: A thicker binding for Lapack Newsgroups: comp.lang.ada

For info, the compiler (well, GNAT) will automatically transpose when doing assignment between arrays with different conventions. And it's quicker (not by very much at -O2, though).

with Ada.Calendar; use Ada.Calendar;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Numerics.Float_Random;
with Interfaces.Fortran;
with System.Generic_Array_Operations;
procedure Sauvage_Timing is

package BLAS is
  -- Copied from old GNAT
  -- Interfaces.Fortran.BLAS.

  -- Vector types
  type Real_Vector is array
    (Integer range <>)
  of Interfaces.Fortran.Real;

  type Complex_Vector is array
    (Integer range <>)
  of Interfaces.Fortran.Complex;

  -- Matrix types
  type Real_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Real;

  type Double_Precision_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Double_Precision;

  type Double_Complex_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Double_Complex;

  type Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Real;

  type Double_Precision_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Double_Precision;

  type Complex_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Complex;

  type Double_Complex_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Double_Complex;
end BLAS;

procedure Transpose is new
    Double_PrecisionBase, Matrix => BLAS.Double_Precision_Matrix);

  type Double_Precision_Matrix is array
    (Integer range <>)
  of Interfaces.Fortran.Double_Precision;

  pragma Convention (Fortran, Double_Precision_Matrix);
A, B : BLAS.Double_Precision_Matrix
(1..100, 1..100);
C : Double_Precision_Matrix (1 .. 100, 1 .. 100);

pragma Volatile (B);
pragma Volatile (C);
Gen :
Start, Finish : Time;
use type Interfaces.Fortran.Double_Precision;
begin
Ada.Numerics.Float_Random.Reset(Gen);
for J in ARange (1) loop
for K in ARange (2) loop
A (J, K) := Interfaces.Fortran.
Double_Precision (J* K) * Interfaces.Fortran.
Double_Precision
Random (Gen));
end loop;
end loop;
Start := Clock;
for J in 1 .. 100 loop
Transpose (A, B);
end loop;
Finish := Clock;
Put_Line ("Transpose took " &
Duration'Image (Finish - Start));
end

A := True;
for J in 1 .. 100 loop
if B (J, K) /= C (K, J) then
Same = False;
end if;
end loop;
for J in 1 .. 100 loop
C := Double_Precision_Matrix (A);
end loop;
Finish := Clock;
Put_Line ("Assignment took " &
Duration'Image (Finish - Start));
end

define array types with components and
indices specified from types defined by
interfaces.fortran.

- uses pragma fortran for vector and
matrix array types.

Evidently conversion between
columnwise and row-wise storage is
handled transparently by Ada language
specification of pragma. Transparent
transposition happens on assignment (but
not on parameter elaboration?). Some
testing is in order here.

I considered enumerating the properties of
System.Generic_Real_LAPACK but if it's
leaving, why bother? –

From: Leo Brown

I'm uncertain about what Ada lapack
materials are in play. Please help me get
the story of the several variations straight.
about System.Generic_Real_LAPACK --
Is it true that this structure that I see in
my gpl 2010 is obsolete? Has it been removed
in later gnats? Has interfaces.fortran.lapack met the same
fate?

About lapack.ads on sourceforge,
attributed to NMA and SJW: --

Is this in fact the only low-level binding
to the Fortran linear algebra subroutines?

Properties of lapack.ads:
- largely auto-generated.
- covers the entire lapack library: single,
double precision, real and complex.
- defines array types with components and
indices specified from types defined by
interfaces.fortran.

- uses pragma fortran for vector and
matrix array types.

- Bush (Business Shell from Pegasoft) for
Linux infra
- Jadam JVM if it works a day x)
- LLVM infra
- AWS,
- Polyorb,
- …

My main goal is to made an OS made by
Ada developers, for Ada developers.
I'd like to see on that OS:
- Polyorb,
- AWS,
- - All known bindings: Gtk, OpenGL, ...
- - Jadvm JVM if it works a day x)
- - LLVM infra
- - Bush (Business Shell from Pegasoft) for
scripting shell
- …

Ahven

From: Tero Koskinen

I've now cleaned up (or more or less re-
written) my stepper motor code. It can be
accessed from:
S git clone http://gustaf.thorslund.org/src/
avr_stepper_motor

Comments are more than welcome.
Ada User Journal Volume 34, Number 1, March 2013

Ada-related Tools

Alven is free software distributed under permissive ISC license and should work with any Ada 95, 2005, or 2012 compiler. Starting from this release, the exception backtraces are now stored to the test results and printed out along with the results. In addition, the documentation received some improvements, and the output of multilime messages from TAP.Runner has been fixed.

- Source text: [2]
- Changelog: [3]


HAC Ada Compiler

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sun, 27 Jan 2013
Subject: HAC Ada Compiler
URL: http://sourceforge.net/projects/hacadacompiler/

HAC - Hacker's Ada Compiler or Hello-world Ada Compiler is meant to be experimental and won't probably ever cover the full language, but an ever growing subset.

It originates from a translation of SmallAda from Pascal to Ada.

HAC is perhaps the first open-source (partial) Ada compiler programmed in Ada itself.

[GNAT is also written (mostly) in Ada. —spare]

GeoTIFF

From: Riccardo Bernardini <framfritti@gmail.com>
Date: Mon, 28 Jan 2013 02:28:39 -0800
Subject: GeoTIFF (TIFF) binding for Ada?
Newsgroups: comp.lang.ada

I need to be able to read the tags in a GeoTIFF file from Ada. On the GeoTIFF page (http://trac.osgeo.org/geotiff/) I found few libraries

GDAL : (C++) http://www.gdal.org/
libgeotiff : (C)
http://www.remotesensing.org/geotiff/api/index.html
libtiff : http://www.remotesensing.org/libtiff/

so I could write a binding for them, but before starting working I wanted to check if there was something ready.

Do you know if there is

1. An Ada binding for libgeotiff/GDAL/ libtiff?
2. An Ada library for reading GeoTIFF tags?

3. An Ada library for reading TIFF tags?
   (GeoTIFF images are just TIFF images with a special set of tags, so that any TIFF-capable library would work)

From: John B. Matthews
Date: Mon, 28 Jan 2013 06:39:09 -0500
Subject: Re: GeoTIFF (TIFF) binding for Ada?
Newsgroups: comp.lang.ada

Indirectly, Ossim, <http://trac.osgeo.org/ossim/> , includes Ossim_Ada,

I've not examined the tags directly.

Ada Server Faces

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Wed, 06 Feb 2013
Subject: Ada Server Faces 0.5.0 is available
URL: http://blog.vacs.fr/index.php/post/2013/02/05/Ada-Server-Faces-0.5.0-is-available

Ada Server Faces [1] is an Ada implementation of several Java standard web frameworks.


Ada Database Objects

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Sun, 10 Feb 2013 22:52:13 +0100
Subject: [Ann] Ada Database Objects 0.4.0 is available
Newsgroups: comp.lang.ada

The Ada Database Objects is an Object Relational Mapping for the Ada 2005 programming language. It allows to map database objects into Ada records and access databases easily. Most of the concepts developed for ADO come from the Java Hibernate (http://www.hibernate.org/) ORM. ADO supports MySQL and SQLite databases.

The new version brings:

- Support to reload query definitions,
- It optimises session factory implementation,
- It allows to customise the MySQL database connection by using MySQL SET.

This version can be downloaded from http://code.google.com/p/ada-ado/downloads/list

Dynamo

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Sun, 10 Feb 2013 23:35:39 +0100
Subject: Dynamo 0.6.0 is available
Newsgroups: comp.lang.ada

Dynamo is a tool to help developers write some types of Ada Applications which use the Ada Server Faces or Ada Database Objects frameworks. Dynamo provides several commands to perform one specific task in the development process: creation of an application, generation of database model, generation of Ada model, creation of database.

The new version of Dynamo provides:

- A new command build-doc to extract some documentation from the sources,
- The generation of MySQL and SQLite schemas from UML models,
- The generation of Ada database mappings from UML models,
- The generation of Ada beans from the UML models,
- A new project template for command line tools using ADO,
- A new distribution command to merge the resource bundles.

The Dynamo tool is available at http://code.google.com/p/ada-gen

Ada Web Application

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Fri, 15 Feb 2013 23:22:56 +0100
Subject: [Ann] Ada Web Application 0.3.0 is available
Newsgroups: comp.lang.ada

Ada Web Application is a framework to build web applications.

- AWA uses Ada Server Faces for the web framework. This framework is using several patterns from the Java world such as Java Server Faces and Java Servlets.
- AWA provides a set of ready to use and extendable modules that are common to many web application. This includes managing the login, authentication, users, permissions.
- AWA uses an Object Relational Mapping that helps in writing Ada applications on top of MySQL or SQLite databases. The ADO framework allows to map database objects into Ada records and access them easily.
- AWA is a model driven engineering framework that allows to design the application data model using UML and generate the corresponding Ada code.

The new version of AWA provides:

- New jobs plugin to manage asynchronous jobs,
- New storage plugin to manage a storage space for application documents,
- New votes plugin to allow voting on items,
- New question plugin to provide a general purpose Q&A.
AWA can be downloaded at http://code.google.com/p/ada-awa/downloads/list

A live demonstration of various features provided by AWA is available at http://demo.vacs.fr/atlas

Yolk

From: Thomas Locke <tl@ada-dk.org>
Date: Thu, 21 Feb 2013
Subject: Updated README to reflect the new SQLite addition to the demo.
URL: https://github.com/ThomasLocke/yolk

Yolk is an Ada based web-toolbox that sits on top of AWS, GNATcoll, Florist and XML/Ada. You will also need to compile GNATcoll with SQLite and/or PostgreSQL support in order to try the Yolk demo. If you don’t want to muck about with the demo, and you have no need for PostgreSQL or SQLite, then you can safely compile GNATcoll without support for these two databases.

It is important to understand that Yolk itself does very little that cannot be accomplished using plain AWS, GNATcoll, XML/Ada and Florist. It is, for lack of a better term, a convenience. Some of things that Yolk offers are:

- Sending emails from the application
- Loading configuration files
- Handling the most common static content types (HTML, PNG, ICO and so on)
- Easy Atom RSS feed generation
- Switching to an un-privileged user.
- Start/stop as a "regular" daemon using an rc script.
- Easy logging to syslogd.

Be sure to check out the Yolk demo application for a good example on how to make use of Yolk.

I've only ever tested Yolk on Linux. I've no idea if it works on Windows or Mac, and I've no current plans for trying my hand at those two platforms. If you'd like to help make it work for Mac and/or Windows, feel free to contact me at <thomas@12boo.net>.

Yolk is GPLv3.

Gate3-in code generator for Glade2 and Glade3

GtkBuilder

From: Rob Groen <robgr@xs4all.nl>
Date: Sat, 23 Feb 2013 16:33:47 +0100
Subject: [gtkada] gate3-in code generator for Glade2 and Glade 3 GtkBuilder and LibGlade files
Mailing list: GtkAda <gtkada@lists.adacore.com>

Inspired by the work of Francois Fabien on a Glade3 code sketcher (see the GtkAda Digest of November 24th 2012) I brought the old gate-in code generator back to life.

The work is based on the sources found in the 2011 GtkAda distribution, notably Glib.Glade, Gik.Glade and Gik_Generates. This is work in progress, so not all Glade3 features are supported (yet). Also, Glade2 testing is limited to Glade2 files that I have used in the past, so they probably don’t touch all the functionality.

Testing has been done on WinXP and on Ubuntu 10.04, using the 2012 GNAT GPL andGtkAda distribution. No other prerequisites are known to me.

Download the zip file from: http://robgr.home.xs4all.nl/

When building gate3-in the gpr file specifies "debug" and "obj" subdirectories relative to the directory where the sources and the gpr file are found.

OpenID authentication for AWS

From: Thomas Locke <tl@ada-dk.org>
Date: Tue, 26 Feb 2013
Subject: openid-client
URL: https://github.com/AdaHeads/openid-client

OpenID authentication for the Ada Web Server library (AWS).

Build (with demo application):

SERVER_NAME=your.server.name make

Run demo application:

sudo ./demo/openid_demo

(This is the OpenID client for AWS, which was mentioned in AUJ 33-4. It has moved to a new location. – sparre)

Ada-related Products

ObjectAda for Windows

From: Atego
Date: Tue, 22 Jan 2013
Subject: Atego Ships Major New Version of Aonix ObjectAda for Windows
URL: http://www.atego.com/pressreleases/pressitem/atego-ships-major-new-aonix-objectada-for-windows-version/

Atego™, the leading independent supplier of industrial-grade, collaborative development tools for engineering complex, mission- and safety-critical architectures, systems, software and hardware, has launched Aonix ObjectAda 9.0 for Windows with new support for Ada 2005.

“Ade continues to be a mainstay in many long-term military & aerospace applications and assuring the tools that support these applications remain not only available, but current and well maintained is critical,” said Dr. Jerry Krasner, Chief Analyst Embedded Market Forecasters (http://www.embeddedforecast.com).

“No only has Ada as a language continued to improve, but the hardware and operating systems have evolved as well. I credit Atego for progressing their tools on both fronts to keep developers well equipped with the latest-and-greatest methodologies and technologies available.” “It’s no mistake that IBM entrusted Atego to take over their Ada line of products.”

Aonix ObjectAda® 9.0 for Windows is a major new release with significant support for many of the new language features in Ada 2005 that users require, as well as a number of new or enhanced features including:

- Aonix ObjectAda operation in either Ada 2005 or Ada 95 mode
- Floating Point Overflow Control
- Non-Ada Threads Support
- Ada Binding to Java Native Interface (JNI)
- File IO Open/Create support for Unicode File Names
- Read/Write support in Text_IO for decoding/Encoding UTF character data
- Runtime Corrections and Enhancements
- Debugger Enhancements for debugging DLLs

This release builds on enhancements provided with the 8.5 release from earlier in 2012. Including enhanced compatibility with Windows 7 and Windows 8 and is based on Microsoft’s Visual Studio 2010 Service Pack 1 development tools and libraries from the Microsoft Windows Software Development Kit (SDK) version 7.1 for Windows 7 and.NET Framework 4. The combination of tools and libraries, included with Aonix ObjectAda v 9.0 for Windows, allows developers to create applications for Windows 7, Server 2008 R2, Server 2008, XPSP3, Vista, and Windows Server 2003 R2. Additionally, the Ada bindings to the Windows API provided by Aonix ObjectAda have been adapted to work correctly with the Windows SDK V7.1 libraries.

“We’re very pleased to announce Ada 05 support in our Aonix ObjectAda for Windows product,” stated Hedley Apperly, Atego’s Vice-President of Product & Marketing. “We are committed to the continued improvement of our products for the Ada language and the long-term availability these products, which respond to our customers’ needs and set them ahead of their competition.”

AWS hosting

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 29 Jan 2013 00:54:06 -0800
Ada User Journal Volume 34, Number 1, March 2013

Subject: Ada Web Server (AWS) hosting
Newsgroups: comp.lang.ada
I'm pleased to announce that Inspirel offers a hosting service for AWS-based projects:
http://inspirel.com/aws-hosting/
The purpose of this service is to promote good practise in web development (which is commonly going against sound software engineering principles) and to foster collaboration between individuals and smaller teams that would like to use Ada for their projects but are frequently limited by the availability of dedicated hosts.

GNAT for ARM
From: AdaCore Press Center
Date: Wed, 27 Feb 2013
Subject: AdaCore Releases GNAT Pro Safety-Critical for ARM Processors
URL: http://www.adacore.com/press/gnat-pro-safety-critical-for-arm/
Ada now available for popular bareboard platform
NEW YORK, PARIS and NUREMBERG, Germany, February 27, 2013 – Embedded World Conference – AdaCore today announced the availability of its GNAT Pro Safety-Critical product for ARM Cortex micro-controllers. This bareboard GNAT Pro Safety-Critical product provides a complete Ada development environment, oriented towards systems that are safety-critical or have stringent memory constraints. Developers of such systems can now exploit the software engineering benefits of the Ada language, including reliability, maintainability, and portability.
ARM processors are becoming more and more prevalent in the aerospace, defense, and transportation industries. This is due in large part to the vibrant support ecosystem that ARM enjoys, and to the growing popularity of these low-cost, low-power microprocessors.
The ARM platform adds to the GNAT Pro Safety-Critical product offering, which is already available for PowerPC and LEON boards, allowing easy portability among all three platforms. The technology does not require any underlying operating system, so it can be deployed on very small memory boards. The tool suite includes the following:
- Support for Ada 2012 (including the important “contract-based programming” features that make it easier to reflect the program’s intent) and all earlier versions of the Ada language.
- Support for the Ravenscar tasking profile.
- A set of static analysis tools:
  o GNATstack analysis tool
  o GNATmetrics complexity metrics tool
  o GNATcheck coding standard verification tool
- The GNATest unit test harness generator.
- The GDB visual debugger.
- A native Integrated Development Environment (IDE) as well as an Eclipse plug-in.
GNAT Pro Safety-Critical for bareboard ARM supplies a fully configurable / customizable run-time library and implements High-Integrity profiles that are especially relevant to safety-critical systems. The Zero Footprint Profile (ZFP) in particular defines an Ada subset that does not require any run-time routines, thus reducing the memory footprint to user code only.
“This new offering shows our commitment to providing a complete safety-oriented development toolset on a large range of targets,” stated Cyrille Comar, Managing Director of AdaCore. “This allows customers to benefit from the richness of the hardware platforms used by the wider market beyond safety-critical systems”
About GNAT Pro Safety-Critical
GNAT Pro Safety-Critical is a complete development environment for applications that need to meet the highest levels of safety-related standards found in industries such as aeronautics, space, railway, defense and medical systems. The product consists of the full GNAT Pro environment enhanced with a suite of tools — specifically GNATcheck, GNATmetric, and GNATstack — and specialized run-time libraries designed for usage in a safety certification context.
Owing to both the product’s technical features and the Ada language’s software engineering foundations, GNAT Pro Safety-Critical facilitates formal compliance with domain-specific safety standards. In addition to supporting RTCA DO-178B / DO-178C (also known as EUROCAE ED-12B / ED-12C), GNAT Pro Safety-Critical can help reduce the effort in certifying systems against standards such as DEF STAN 00-55 / 00-56 (defense), DO-278 / DO-278A (ground-based systems), CENELEC EN 50128 (rail) and ECSS-E-ST-40C / ECSS-Q-ST-80C (space). GNAT Pro Safety-Critical has been used to develop systems that have been certified to DO-178B Level A.

Ada and GNU/Linux
Debian
From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sun, 30 Dec 2012 10:10:15 +0100
Subject: Debian GNAT 4.7, status?
Newsgroups: comp.lang.ada
Any soon?
P.S. I am asking because yesterday I discovered a severe bug in Debian’s GNAT 4.6 (compiler freeze). Fedora is already 4.7 and there is no bug.
From: Ludovic Brenot <ludovic@ladovic-brenot.org>
Date: Sun, 30 Dec 2012 14:51:00 +0100
Subject: Re: Debian GNAT 4.7, status?
Newsgroups: comp.lang.ad
I uploaded 4.7.2-1~exp1 to experimental (which you must install on top of unstable) a couple of days ago. The packages are still in the NEW queue pending manual review and approval. And because this is experimental, this compiler is for experimental purposes only and no libraries, besides libgnat, built with it.
I don’t think I’ll spend much time working on 4.7, Debian 8 “Jessie” will probably have 4.8 (or later) instead.

Ada and Microsoft
Compiler for Windows7
From: Vernon Marsden
<vmars316@gmail.com>
Date: Fri, 4 Jan 2013 15:00:10 -0800
Subject: where can I find a freeware ADA compiler for home use (windows7) ?
Newsgroups: comp.lang.ada
Where can I find a freeware Ada compiler for home use (Windows7)?
Also, is there a visual IDE that goes with Ada?
From: Britt <britt.snodgrass@gmail.com>
Date: Fri, 4 Jan 2013 19:45:32 -0800
Subject: Re: where can I find a freeware ADA compiler for home use (windows7) ?
Newsgroups: comp.lang.ada
http://libre.adacore.com/ is the place to start.
From: Stephen Leake
<stephen_leake@stephe-leake.org>
Date: Sat, 5 Jan 2013 09:17:27 -0500
Subject: Re: where can I find a freeware ADA compiler for home use (windows7) ?
Newsgroups: comp.lang.ada
GPS comes with the GNAT GPL package, and Emacs has an Ada mode (currently being upgraded; see http://stephe-leake.org/emacs/adamode/adamode.html)
From: Dennis Lee Bieber
<dlieber@ix.netcom.com>
Date: Sat, 5 Jan 2013 14:09:53 -0500
Subject: Re: where can I find a freeware ADA compiler for home use (windows7) ?
Newsgroups: comp.lang.ada

[45x608]2. Download Adagide at
http://sourceforge.net/projects/adagide/.

References to Publications

Writing bindings for C libraries

From: Felix Krause
Date: Wed, 13 Jun 2012
Subject: Writing Ada Bindings for C Libraries, Part 1
URL: http://flyx.org/2012/06/13/adabinbounds1/

This article gives an overview over problems, solutions and guidelines for writing an Ada binding for a C library. It summarises experiences I made while implementing OpenGLAda and OpenGLAd. Code examples are taken from those projects.

The Ada code examples shown here are written in Ada 2005. Note that you can import C functions somewhat nicer in Ada 2012.

[The last of three parts was published February 18th. —sparre]

Ada on Raspberry Pi: new article in MagPi issue 8

From: Rego, P. <pvrego@gmail.com>
Date: Mon, 3 Dec 2012 17:43:50 -0800
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsgrups: comp.lang.ada

> Any ideas? [...] 5) Maybe tasking

Tasking is one of Ada's excellent features, IMO. If we are going to introduce kids to programming via Ada, it might well behoove us to spoil them on parallel-computing w/ tasks.

> Any ideas? [...] Might I suggest an interpreter? You could explain it a bit more succinctly [and with less jargon (no need to go into linking, object files, loaders, etc)] than a compiler. (To keep syntax & parsing simple you could go with the old-school/early-computing LISP.)

From: Simon Wright
Date: Mon, 3 Dec 2012 17:43:50 -0800
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsgrups: comp.lang.ada

> 4) Representation clauses for mapping to hardware or C APIs Based on what John McCormick found to be so valuable about Ada's expressive fundamental type system, I'd like to humbly suggest that it be made known, and preferred to implementation defined C like types such as Integer. In particular, Ada's scalar types allow programmers to just say what they already know instead of thinking hard about which int-like type might somehow match what they cannot express in the fundamental type system (of C, or Ada's library types), such as range 1 .. 80,000 (vs C's INT_MAX/Ada's System.Max_Int):


From: Simon Wright
Date: Mon, 3 Dec 2012 17:43:50 -0800
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsgrups: comp.lang.ada

> Any ideas? [...] I think that sometimes it's a good incentive (for people to use a programming language) to have some device libraries implemented, so they can use the Ada features, but don't have to implement all of it from scratch. Maybe GPIO? A good play could be to demonstrate the tasking/rendezvous/semaphores concepts using led's through GPIO. Raspberry is a good portfolio for motivate more people to use Ada, and your articles follows it. Very good.

From: Simon Wright
Date: Mon, 3 Dec 2012 17:43:50 -0800
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsgrups: comp.lang.ada

> Any ideas? [...]
The one I'm having fun with is an interface to I2C, targeted at the MCP23017 I/O expander.

"Writeup": http://raspi-i2c-ada.sourceforge.net/

Code tip: https://sourceforge.net/p/raspi-i2cada/code/ci/e85f5cbe2960a7f306dced
917d8d79e61d7c6399/tree/

At the moment it detects input changes via polling; I'm working on an interrupt-driven variant (the nearest to interrupt-driven you'll get in userland, anyway, I think) but I haven't pushed those changes yet.

The "interrupt" changes will be based on WiringPi at https://projects.drogon.net/raspberry-pi/wiringpi/ and will effectively provide an Ada interface to GPIO - I'm trying to avoid too much direct work with GPIO in case I let the magic smoke out.

Work is a tad held up due to illness; hopefully both I and the work are on the road to recovery!

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>

Date: Tue, 11 Dec 2012 12:04:35 +0000
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsroups: comp.lang.ada

> Any ideas? [...]

For the "cool factor" (but maybe too large otherwise) one of my colleagues has an Ada managed phone system running on a Raspberry Pi. (You can call his phone queue on +45 77 34 34 20, but nobody is answering yet. :-(

The source code for the Ada part will be available on GitHub.

From: Simon Wright <simon@pushface.org>

Date: Tue, 11 Dec 2012 12:34:23 +0000
Subject: Re: Ada on Raspberry Pi: new article in MagPi issue 8
Newsroups: comp.lang.ada

I see I've already got some early code up there: [1].

The reason for going with an interface like wiringPi, by the way, is that direct access to GPIO requires root privilege. wiringPi provides a setuid-root program gpio; it lets you - for example - access the value on GPIO pin x at /sys/class/gpio/gpio[x]/value.

[1] https://sourceforge.net/u/simonjwright/wiringpi-ada/code

**OOP in embedded critical systems**

From: Quentin Ochem
Subject: Object Orientation in Embedded Critical Systems – don’t be scared anymore!
Date: Fri, 07 Dec 2012


Object Oriented Programming (OOP) was introduced to software development almost half a century ago, and its popularity has reached all fields of software development. All? Well almost. OOP was banished from embedded, safety-critical development for this entire time, so many of the latest design methodologies often based on these patterns became wishful thinking.

 […]

**Top ten milestones in embedded 2012**

From: Jack Ganssle <jack@ganssle.com>
Date: Wed, 19 Dec 2012
Subject: Top ten milestones in embedded 2012

Number 9: Ada 2012

 […]

**Heart Pump Software with SPARK and Echo**

From: John Knight, Xiang Yin and Patrick John Graydon
Date: Tue, 01 Jan 2013
Subject: Formally Verifying Heart Pump Software with SPARK and Echo
URL: http://www.medicaldesignbriefs.com/component/content/article/15336

The Ada-based SPARK programming language and toolset offer strong guarantees about the behaviour of software systems. This powerful core underpins Echo, a complete approach to practical formal verification developed at the University of Virginia, Charlottesville, by Dr. John Knight and his student, Xiang Yin. Echo extends the verification power afforded by the SPARK tools and the PVS Specification and Verification System to provide a complete proof that a given SPARK implementation re fincs its formal specification.

 […]

**AdaCore answers questions from Jack Ganssle**

From: Jack Ganssle <jack@ganssle.com>
Date: Mon, 14 Jan 2013
Subject: Ada 2012 redux

In September I wrote about the new Ada 2012 standard ("Ada gets a makeover"). That article elicited quite a few responses, and some misinformation. So, I digested some of the discussion into questions, which I sent to the folks at AdaCore. They were kind to reply. Here's that discussion.

[Good questions, debatable answers. — sparre]

**Contract-Driven Programming Takes Specification Beyond The Stone Age**

From: Quentin Ochem
Date: Mon, 04 Feb 2013
Subject: Contract-Driven Programming Takes Specification Beyond The Stone Age

The industrialisation age of programming by contract is opening a new era in software development. Just as development techniques went from assembly to structured languages and from structured languages to object orientation, contract-based programming is providing one more abstraction to software design.

 […]

Java specifies what exceptions may be thrown, Ada enables you to specify type ranges, C allows… um, never mind.

[The paper covers JML (Java), ACSL (C) and Ada with a few references to SPARK thrown in for good measure. — sparre]

**Reducing the gap between design and code for critical software with Ada 2012**

From: S. Tucker Taft, AdaCore
Date: Mon, 14 Feb 2013
Subject: Reducing the gap between design and code for critical software with Ada 2012

The latest version of the Ada language standard, known as Ada 2012 [1], has extended the language's support for minimising the gap between design and code by raising the level of discourse, a principle that is fundamental to software development and that has been an underlying goal throughout Ada's evolution.

 […]

**Valentines Greetings**

From: AdaCore Press Center
Date: Mon, 14 Feb 2013
Subject: Ada Passion: Developers speak out about why they use Ada and what they love about it.
URL: http://www.ada2012.org/passion.html

[Unfortunately we can't reprint the nice video here. — sparre]

[...] “Ada is readable and reliable. She's a hard mistress that keeps me from doing bad things, and coming from PHP I know all about doing bad things. With her strong emphasis on types and structure Ada keeps me on the right path, constantly helping me to better express what it is I'm trying to accomplish with my code.” — Thomas Lacké

“Good practise from the ground, I like Ada as she talks to me with the right abstraction level. Thanks Ada, we understand each other.” — Pascal Obry

[... and lots of other quotes about why developers like Ada. —sparre]

**Embedded Newsletter**

From: UBm Tech
Date: Tue, 19 Feb 2013
URL: http://e.ubmektronics.com/audience/etnnewletters/02-19-2013-EmbeddedNL.html

Tech Focus: Using Ada in embedded design

- Is Ada ready for mainstream embedded designs?
- Reducing the gap between design and code for critical software with Ada 2012
- Ada 2012: say what you mean, mean what you say
- Ada 2012 redux

[An embedded newsletter with a strong focus on Ada. —sparre]

**Ada Inside**

**Traceability Analysis Helps Rockwell Collins Achieve DO-178B Certification**

From: AdaCore Press Center
Date: Tue, 04 Dec 2012
Subject: Traceability Analysis Helps Rockwell Collins Achieve DO-178B Certification
URL: http://www.adacore.com/press/traceability-analysis/

BOSTON, December 4, 2012 - ACM SIGAda HILT Conference - AdaCore today announced the successful usage of its Code Traceability Analysis for DO-178B by Rockwell Collins in the certification of the Integrated Display System (IDS) for a large, next-generation, commercial aircraft. The Traceability Analysis package is part of the evidence needed to satisfy the DO-178B objectives for structural code coverage at Level A, the highest (most stringent) level for avionics software safety.

The IDS, featuring Rockwell Collins’ EFIS/EICAS Interface Unit (EIU-7001), includes many of the advanced features found on the Boeing 787, such as an electronic checklist with cursor control panel, navigation performance scales and vertical situation displays. This critical system is Level A and was developed in Ada using AdaCore’s GNAT Pro High- Integrity Edition for DO-178B, targeted to PowerPC-ELF and using the Zero Footprint (ZFP) run-time library.

Certification at Level A is a major effort, involving a variety of software life cycle processes. One of the verification process activities entails demonstration of complete coverage of the source code through requirements-based tests. If any object code is not traceable to the source code (for example, if a high-level source construct is compiled into complex object code involving conditional instructions) then DO-178B specifies the following activity: ‘additional verification should be performed on the object code to establish the correctness of such generated code sequences’. AdaCore has developed an infrastructure and methodology to perform this activity; the product of this activity is a source-to-object Code Traceability Analysis.

AdaCore’s approach has been successfully used in the past, saving customers time and effort in conducting DO-178B certification. In light of this successful previous experience, Rockwell selected AdaCore to conduct an analogous traceability analysis for this next generation IDS Program. Rockwell has now successfully completed the certification of its IDS, with AdaCore’s Traceability Study serving as part of the full certification evidence.

“ traceability analysis that is needed to meet the structural code coverage objectives requires a detailed knowledge of both the source language semantics and the generated object code,” said Robert Dewar, AdaCore President and CEO.

“With our expertise in the Ada language and compiler technology, AdaCore was uniquely qualified to perform this analysis both accurately and efficiently. This allowed Rockwell Collins engineers to devote their energies to certification activities directly related to their actual application.”

**Atmosphere Space Interactions Monitor**

From: AdaCore Press Center
Date: Mon, 17 Dec 2012
Subject: GNAT Pro Safety-Critical used by Terma A/S for Space Monitor Project
URL: http://www.adacore.com/press/termar

PARIS, NEW YORK, December 17, 2012 - Paris Space Week 2012- AdaCore today announced that Terma A/S has selected the GNAT Pro Safety-Critical development environment to develop onboard software for the Atmosphere Space Interactions Monitor (ASIM) that will be mounted on the Columbus module of the International Space Station. Terma will use GNAT Pro Safety-Critical combined with the GNATemulator and GNATcoverage dynamic testing tools to develop and test the application prior to deployment on the actual LEON 3 embedded processor.

ASIM is used to detect lightning formations known as “red sprites”, “blue jets” and “elves”, and to detect X-ray and γ-ray discharges. The objective is to search for a correlation between these formations and large thunderstorms, improving our understanding of these phenomena and their influence on the Earth’s climate. Terma is – under contract to the European Space Agency (ESA) – the prime contractor for the development of the ASIM instrument including development of the on-board software. ASIM will be deployed in space where repairs are costly if possible at all, making reliability of the platform and its software essential. This need for reliability was a principal factor in selecting the Ada programming language for the software development. The Ravenscar profile (a subset of the Ada tasking features designed for safety-critical hard real-time computing) will be used to ensure that all multi-processing/tasking within the application can be proven deterministic and schedulable. Ada’s ability to define static and dynamic contracts and checks – including features recently introduced in the new Ada 2012 standard – helps developers express requirements directly in the software. This allows early detection of inconsistencies, either statically (at compile time) or dynamically (during testing).

To carry out the Ada development, Terma selected the LEON 3 ELF configuration of the GNAT Pro Safety-Critical development environment. It includes tools that take advantage of the language’s properties to perform additional static and dynamic analysis, reaching even higher levels of reliability. Complexity and other metrics are automatically monitored using GNATmetrics, while GNATcheck enforces a consistent coding style, and detects well-defined categories of code.

Volume 34, Number 1, March 2013
Ada in Context

IBM 437 encoded String to UTF-16 Wide_String

From: Gautier de Montmollin
<gaudet.de.montmollin@gmail.com>
Date: Tue, 27 Nov 2012 13:02:05 -0800
Subject: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Type ? I'm too lazy for that :-). I've put this [from the Wiki page] into Excel, and it's a bit of a hassle. I've put this into a string type.

AdaChess

From: Alessandro Iavicoli
Date: Tue, 22 Jan 2013
Subject: AdaChess
URL: http://www.alessandroiavicoli.it/

AdaChess is a simple, small yet complete chess engine written in Ada. The code is clean, well-formed and readable. Talking in terms of chess engine programming, AdaChess comes with a lot of features! AdaChess has been developed using easy-to-do data structures. Chessboard is represented with an array of 64 elements. Each piece can move inside the board using a piece-offset table. The move-generator is a resource-heavy since it looks only for legal moves before "register" a move as a valid move. This means that the generator is slow but he returns a vector with only the playable, legal moves.

In the Ada 2012 standard (I found the very useful UTF-8 <-> UTF-16 conversions)? In GNAT? In some open-source package?


From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Tue, 27 Nov 2012 22:38:05 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

That doesn't look possible, since IBM 437 is a character set (a mapping from code points to glyphs) while UTF-16 is an encoding scheme (a way to compress all 10646 code points on 16 (or sometimes 32) bits values.

For an explanation of these strange terms, refer to the discussion section of AI05-0137-2/03.

From: Gautier de Montmollin <gaudet.de.montmollin@gmail.com>
Date: Tue, 27 Nov 2012 14:12:04 -0800
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

That looks possible, since IBM 437 is a character set (a mapping from code points to glyphs) while UTF-16 is an encoding scheme (a way to compress all 10646 code points on 16 (or sometimes 32) bits values.

For an explanation of these strange terms, refer to the discussion section of AI05-0137-2/03.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 27 Nov 2012 15:13:00 -0800
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

That looks possible, since IBM 437 is a character set (a mapping from code points to glyphs) while UTF-16 is an encoding scheme (a way to compress all 10646 code points on 16 (or sometimes 32) bits values.

For an explanation of these strange terms, refer to the discussion section of AI05-0137-2/03.

From: Dmitry A. Kazakov <mailto:dmitry-kazakov.de>
Date: Tue, 27 Nov 2012 23:14:22 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

That looks possible, since IBM 437 is a character set (a mapping from code points to glyphs) while UTF-16 is an encoding scheme (a way to compress all 10646 code points on 16 (or sometimes 32) bits values.

For an explanation of these strange terms, refer to the discussion section of AI05-0137-2/03.

Ada in Context
In case of differentiation by sets of code points, I'd rather have an honest type Unicode_String and—if we are already fixing the language—put everything that has {Wide_}String in its name in Annex J.

But then, consider

```adastyle
| type Index is range 1 .. 12;   |
| type R is ("I", "V", "X", "L", "C", "D", "M"); |
| type N is array (Index range <=) of R; |
```

A string of R, named N here, is just fine. In fact,

```adastyle
Year : constant N := "MCMLXXXIII"; |
```

has a valid literal, and the year so written is not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 15:23:08 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

It says that instance implementing the interface are substitutable where a string is expected. You should be able to pass IBM_437_String to Put_Line, Trim, To_Lower etc.

From: Randy Brukardt
<brandy@rrsoftware.com>

Date: Wed, 28 Nov 2012 21:18:23 -0600
Subject: Re: Ada.Directories and UTF-8 encoding
Newsgroups: comp.lang.ada

It's not of any of the standard string types. The definition of type R actually implies a codespace, and, for example, Character("V") or Wide_Character("V") have no role in it, irrespective of any accidental overlap in encoding or representation or position.

So, which by force should type N be in Whatever_String'Class?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>

Date: Wed, 28 Nov 2012 14:36:02 +0100
Subject: Re: IBM 437 encoded String to UTF-16 Wide_String
Newsgroups: comp.lang.ada

Per inheritance:

```adastyle
| type N is new Wide_Wide_String and array (...) of R; |
```
Fun with predicates

From: Bill Findlay
Date: Thu, 6 Dec 2012 09:56:17 -0800
Subject: Re: Press Release - Ada 2012
Newsgroups: comp.lang.ada

I raised this issue a while ago. Using UTF-8 encoded names appears to work with the current packages (on my Linux system) but the Ada standard really needs to be extended to have Wide_versions of various file, directory, etc. packages.

Fun with predicates

If n < 2 then return False; end if;
for d in candidate range 2..3 loop

procedure pn is

begin

function is_prime (nr : candidate) return Boolean is
  n : constant candidate := nr;
  j : candidate;

begin

end if;
end loop;

...
I like my IDE (Emacs) to capitalise for me, and this would need so many case exceptions that I’d become impractical. I have a similar trouble with GPS identifier completion, to only with standard Ada and ‘external/contributed’ packages.

From: Rod Kay
<rodakay@internode.on.net>

Subject: Re: On Style: Any opinions on *not-capitalising the first letter of verbs in a procedure name ?
Newsgroups: comp.lang.ada

Breaking with convention has been the main criticism I’ve received from others so far. I agree it’s definitely a ‘con’.

From: Britt <britt.snodgrass@gmail.com>
<Date: Fri, 4 Jan 2013 20:19:45 -0800
Subject: Re: On Style: Any opinions on *not-capitalising the first letter of verbs in a procedure name ?
Newsgroups: comp.lang.ada

I agree with Bob Dufl’s reply. I think anything starting with lowercase (other than keywords) looks weird and would distract me. The common Ada convention is to capitalise first letters and each letter immediately following an underscore. I also prefer to capitalise the first letter of attributes as in ‘Pos’. Acronyms should be all caps.

Your proposal would be awkward for most pretty printers like to handle but I think gnapp could support it using its default "as declared" casing mode. Gnapp also supports casing exception dictionaries.

From: Randy Brukardt
<randy@rrsoftware.com>
<Date: Fri, 4 Jan 2013 23:13:41 -0600
Subject: Re: On Style: Any opinions on *not-capitalising the first letter of verbs in a procedure name ?
Newsgroups: comp.lang.ada

> Your proposal would be awkward for most pretty printers like to handle but I think gnapp could support it using its default “as declared” casing mode. Gnapp also supports casing exception dictionaries.

From: Britt <britt.snodgrass@gmail.com>
<Date: Fri, 4 Jan 2013 14:30:42 -0800
Subject: Re: On Style: Any opinions on *not-capitalising the first letter of verbs in a procedure name ?
Newsgroups: comp.lang.ada

That makes sense, other than "what everybody else does" is not well-defined. There still seems to be a lot of code in the Ada 83 style around, just to take one example.

At RR Software, our style guide recommends using "Title Case" for identifiers. That is, capitalise an identifier the same way as you would a title. (Our pretty printer will automatically use this style if requested.)

For instance, taking some examples from one of the AI tools:

procedure Put_AI_Header_and_Subject
(Fyle : in Ada.Text.IO.File_Type;
   Issue : in AI_Types.AI_Identifier) is
   -- Put a line containing an AI identifier, -- version, last modification date, and -- subject to Fyle.

procedure Records_of_Response
(TXT_Name, RTF_Name, HTML_Name : in String);

Page 20
After thinking about why your proposal looks bad to me I realised that I always regard the name of a subprogram as a singular proper noun (rather than a sentence in itself) because it is used in the source code as a proper name. E.g.

```ada
procedure Run_Very_Quickly
  (Direction : in Direction_Type) is ...
and, at the point of use:
  Run_Very_Quickly (Direction => North By Northwest);
```

it is the name of the procedure and should be capitalised as a name (proper noun).

From: John English <john.foreign@gmail.com>

Date: Fri, 11 Jan 2013 00:01:20 -0800

Subject: Re: On Style: Any opinions on *not- capitalising the first letter of verbs in a procedure name ?

Newsgroups: comp.lang.ada

So perhaps you should ask yourself these questions:

- Is_Is_Easier_To_Read?
- Will_Autoformatting_Still_Work?

and then decide to either do_It or Avoid_It… :)?

State machines

From: david.mentre@gmail.com

Date: Mon, 21 Jan 2013 02:42:17 -0800

Subject: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

Would somebody have references on the encoding of State Machines or Hierarchical State Machines in Ada?

I am looking for best practices for such encoding (reasonable efficiency, safe encoding) or pointer to publicly available reference code. The application domain is safety critical systems with real time constraints (timers).

For now, I have found "Implementation of state machines with tasks and protected objects" (Ada User Journal 20:4 (Jan 2000), 273-288).

Moreover, I am strongly interested in the formal verification of such machines using SPARK or GNATprove tools. So if somebody is aware of research work on automata in Ada with such tools, I would be very interested in them.

From: Rolf Ebert <rrr.eee.27@gmail.com>

Date: Wed, 13 Feb 2013 08:02:23 -0800

Subject: Re: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

I recommend looking at "Rhapsody in Ada", now by IBM, formerly iLogix. I used it around 2004. AFAIK it is still available.

"Rhapsody in Ada" is a UML tool very strong in (hierarchical) state machines. There are external add-ons (provers) for almost everything you can prove in a state machine.

I did not consider the generated Ada code as very efficient, though (don't remember why).

From: Simon Wright <simon@pushface.org>

Date: Wed, 13 Feb 2013 18:05:57 +0000

Subject: Re: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

There was a paper at an Ada UK conference a while back - the idea was machine verification of hand-coded state charts; turned out that the implementers didn't understand the state machines the way the designers had intended, and they got them wrong anyway.

So I'd be reluctant to see hierarchical state machines; also tricky to reason about.

From: Jeffrey Carter <jrcarter@acm.org>

Date: Wed, 13 Feb 2013 13:25:43 -0700

Subject: Re: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

I recall someone named Robert Dewar (whoever he is :) saying that the mechanical implementation of STDs is a place where one should use "goto". That was here on c.l.a:

https://groups.google.com/forum/
  ?fromgroups=Rtopic.comp.lang.ada/
  tFlS5d7bfpo

This might also be interesting:

https://groups.google.com/forum/
  ?fromgroups=Rtopic.comp.lang.ada/
  FPegCSWstXk1-25-false]

From: Simon Wright <simon@pushface.org>

Date: Wed, 13 Feb 2013 22:28:40 +0000

Subject: Re: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

I can understand that if the FSM is a task - you're implementing the states in terms of the program counter, nothing wrong with goto for that. But if you've inverted this, so that the FSM is a passive construct, you're going to need an enumeration of the possible states, and some way of representing the possible events (I made an abstract type Event_Base with an abstract Handler, then each concrete event that could be received by a particular FSM has a case statement over the possible states; the whole thing generated from a UML model [1]).


coldframe/events.html

From: jwoodruff@gmail.com

Date: Wed, 13 Feb 2013 16:00:12 -0800

Subject: Re: References on encoding (Hierarchical) State Machines / Automata in Ada?

Newsgroups: comp.lang.ada

I wonder if you are aware of FSMedit by Christoph Grein, dated about 2004.

http://www.christ-usch-grein.homepage.t-online.de/Ada/FSM.html

This is plainly not directly applicable to "safety critical systems with real time constraints". However perhaps some useful thoughts.

Ada up on the transparent language popularity index

From: Thomas Lacke <tl@ada-dk.org>

Date: Sun, 17 Feb 2013

Subject: Ada has moved one up at the transparent language popularity index for February 2013!

URL: https://plus.google.com/112815230781381920/posts/ZdKfFo8i9o3

She's now sitting at #8 for compiled languages. There's still a good way to go before we overtake Pascal, but with a concerted effort I'm sure we can make it happen. :)?

http://lang-index.sourceforge.net
Conference Calendar

Dirk Craeynest
KU Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

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

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

2013

April 08-11  25th IEEE Software Technology Conference (STC'2013). Salt Lake City, Utah, USA. Theme: "Back to the Future: Enabling Mission Success through Software Technology". Topics include: cybersecurity, software production efficiencies, resilient software, software engineering best practices, etc.


May 06-08  15th Workshop Software Reengineering (WSR'2013). Bad-Honnef, Germany. Topics include: all activities that aim at examination and alteration of a software system to reconstitute it in a new form, such as: methods and models for software reengineering; embedding reengineering activities into software development; tools for program comprehension, redocumentation, software visualization, static and dynamic program analysis, software test, software maintenance, program transformation; interoperability between software reengineering and development tools; migration strategies and transformation approaches; architecture reconstruction, assessment, migration; approaches for assessing software quality; improving software quality by reengineering; economical aspects of reengineering; experience reports about maintenance activities, migration and reengineering projects; etc.

May 14-16  5th NASA Formal Methods Symposium (NFM'2013). Moffett Field, California, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; formal verification, including theorem proving, model checking, and static analysis; techniques and algorithms for scaling formal methods, including but not restricted to abstraction and symbolic methods, compositional techniques, as well as parallel and distributed techniques; model-based development; applications of formal methods to aerospace systems; formal methods for multi-core implementations; etc.

♦ May 14-17  DAta Systems In Aerospace (DASIA'2013). Porto, Portugal.

♦ May 18-26  35th International Conference on Software Engineering (ICSE'2013). San Francisco, USA. Theme: "Software Engineering Ideas to Change the World".

May 18-19  10th Working Conference on Mining Software Repositories (MSR'2013). Topics include: mining of repositories across multiple projects; characterization, classification, and prediction of software defects based on analysis of software repositories; techniques to model reliability and defect occurrences; search techniques to assist developers in finding suitable components and code fragments for reuse; empirical studies on extracting data from repositories of large long-lived and/or industrial projects; mining execution traces and logs; etc.

May 19-21  26th Conference on Software Engineering Education and Training (CSEET'2013). Theme: "SE Education and Training: maintaining quality in an uncertain future". Topics include: modern development methods, and particularly agile or lean methods; tools and
environments, including open source and commercial products; industry-academia collaboration; technology in support of education and training; etc.

May 20-24 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'2013). Boston-Cambridge, Massachusetts, USA. Topics include: all areas of parallel and distributed processing, such as parallel and distributed algorithms, applications of parallel and distributed computing, parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, and programming environments and tools, etc.

May 24 14th International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-13). Topics include: parallel and distributed computing techniques and codes; practical experiences using various parallel and distributed systems; loop and task parallelism; scheduling and load balancing; compiler issues for scientific and engineering computing; scientific and engineering computing on parallel computers, multicores, GPUs, FPGAs, ...; etc.

June 03-06 8th International Federated Conferences on Distributed Computing Techniques (DisCoTec'2013). Firenze, Italy. Includes the COORDINATION, DAIS, and FMOODS/FORTE conferences. Deadline for early registration: April 25, 2013.

June 03-05 13th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'2013). Topics include: all aspects of distributed applications and systems, throughout their lifecycle; design, architecture, implementation and operation of distributed computing systems, their supporting middleware, appropriate software engineering methods and tools, as well as experimental studies and practical reports; language-based approaches; domain-specific languages; etc.

June 04-06 13th International Conference on Software Process Improvement and Capability Determination (SPICE'2013). Bremen, Germany. Topics include: process assessment, improvement and risk determination in areas of application such as automotive systems and software, aerospace systems and software, medical device systems and software, safety-related systems and software, financial institutions and banks, small and very small enterprises, etc.

June 04-07 22nd Australasian Software Engineering Conference (ASWEC'2013). Melbourne, Australia. Topics include: empirical research in software engineering; formal methods; legacy systems and software maintenance; measurement, metrics, experimentation; modularisation techniques, including component-based software engineering and aspect-oriented programming; programming techniques, such as object-oriented, functional and hybrid programming; open source software development; quality assurance; real-time and embedded software; software analysis; software design and patterns; software engineering education; software processes and quality; software re-use and product development; software risk management; software security, safety and reliability; software verification and validation; standards and legal issues; etc.

June 10-14 18th International Conference on Reliable Software Technologies - Ada-Europe’2013. Berlin, Germany. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN.

June 10-14 10th International Conference on integrated Formal Methods (iFM’2013). Turku, Finland. Topics include: the combination of (formal and semi-formal) methods for system development, regarding modeling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

June 12-14 14th International Conference on Product Focused Software Development and Process Improvement (PROFES'2013). Paphos, Cyprus. Topics include: software engineering techniques, methods, and technologies for product-focused software development and process improvement as well as their practical application in an industrial setting.

June 16-21 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'2013). Seattle, Washington, USA. Topics include: programming languages, their design, implementation, development, and use; innovative and creative approaches to compile-time and runtime technology, novel language designs and features, and results from implementations; language designs and extensions; static and dynamic analysis of programs; domain-specific languages and tools; type systems and program logics; checking or improving the security or correctness of programs; memory management; parallelism, both implicit and explicit; debugging techniques and tools; etc.
June 16-23  ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'2013). Topics include: programming language challenges (features to exploit multicore architectures; features for distributed and real-time control embedded systems; language features and techniques to enhance reliability, verifiability, and security; virtual machines, concurrency, inter-processor synchronization, and memory management; ...); compiler challenges (interaction between embedded architectures, operating systems, and compilers; support for enhanced programmer productivity; support for enhanced debugging, profiling, and exception/interrupt handling; ...); tools for analysis, specification, design, and implementation (distributed real-time control, system integration and testing, run-time system support for embedded systems, support for system security and system-level reliability, ...); etc.

June 17-21  4th International Symposium on Architecting Critical Systems (ISARCS'2013). Vancouver, Canada. Topics include: tools for construction, design, and testing of critical systems; architectural description languages, models, and notations; evolution and maintenance of critical systems; industrial reports, case studies, and application domains; formal methods, model checking, and theorem proving for critical software design; etc.

June 18-21  13th International Conference on Software Reuse (ICSR'2013). Pisa, Italy. Theme: "Safe and Secure Reuse". Topics include: guaranteeing safety and security related properties of reusable components, certification issues for mission-critical reusable components, component-based reuse, COTS-based development, generator-based techniques, domain-specific languages, testing in the context of software reuse, model-driven development, reuse of non-code artifacts (process, experience, etc.), software product line techniques, quality-aspects of reuse, industrial experience with reuse, software evolution and reuse, large-scale systems, etc.

June 19-21  16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC'2013). Paderborn, Germany. Topics include: Programming and system engineering (ORC paradigms, languages, model-driven development of high integrity applications, specification, design, verification, validation, testing, maintenance, ...); System software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...); Applications (embedded systems (automotive, avionics, consumer electronics, ...), real-time object-oriented simulations, ...); System evaluation (timeliness, worst-case execution time, dependability, end-to-end QoS, fault detection and recovery time, ...); etc.

June 20-21  Symposium on Languages, Applications and Technologies (SLATE'2013). Porto, Portugal. Topics include: programming language concepts and methodologies; design of novel language constructs and their implementation; Domain Specific Languages design and implementation; programming tools; programming, refactoring and debugging environments; dynamic and static analysis: program slicing; compilation and interpretation techniques; code generation and optimization; etc.


June 26-28  16th International System Design Languages Forum (SDL'2013). Montréal, Canada. Topics include: dependability (availability and reliability) engineering, approaches and standards; industrial application reports (industrial usage reports, standardization activities, tool support and frameworks, domain-specific applicability such as telecommunications, aerospace, automotive, control, etc.); evolution of development languages: domain-specific language profiles especially for dependability, modular language design, semantics and evaluation, methodology for application; etc. Deadline for submissions: May 15, 2013 (posters, exhibit proposals).


June 27-30  12th International Symposium on Parallel and Distributed Computing (ISPDC'2013). Bucharest, Romania. Topics include: tools and environments for parallel program analysis, parallel programming paradigms and APIs, distributed systems methodology and networking, distributed software components, task scheduling and load balancing, fault tolerance in parallel and distributed systems, security in parallel and distributed systems, performance management in parallel and distributed systems, real-time distributed and parallel systems, etc. Deadline for early registration: April 25, 2013.

July 01-03 18th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE’2013). Canterbury, Kent, UK.

July 01-03 7th International Symposium on Theoretical Aspects of Software Engineering (TASE’2013). Birmingham, UK. Topics include: the theoretical aspects of model driven software engineering, component based software engineering, software security, reliability, and verification, embedded and real time software systems, aspect and object oriented software design, reverse engineering, etc.

 широко July 01-05 27th European Conference on Object-Oriented Programming (ECOOP’2013). Montpellier, France. Topics include: all areas of object technology and related software development technologies, such as aspects, components, modularity, concurrent and parallel systems, distributed computing, programming environments, versioning, refactoring, software evolution, language definition and design, language implementation, compiler construction, design methods and design patterns, real-time systems, security, specification, verification, type systems, etc. Deadline for submissions: April 5, 2013 (tutorials), April 15, 2013 (research project symposium abstracts). Deadline for early registration: June 1, 2013.

 широко July 01 Workshop on Mechanisms for SPEcialization, Generalization and inHerItance (MASPEGHI’2013). Topics include: the design of inheritance-related reuse mechanisms, including their dynamic semantics, static analysis, permissions and visibility; software engineering issues, including metrics, interactions with methodologies, and consequences for quality parameters such as maintainability and comprehensibility. Deadline for paper submissions: April 21, 2013.

 широко July 02 8th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS’2013). Topics include: efficient implementation and compilation of OO languages in various application domains ranging from embedded and real-time systems to desktop systems. Deadline for paper submissions: April 19, 2013.

 широко July 03-07 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE’2013). Angers, France. Topics include: emerging as well as established SE methods, practices, architectures, technologies and tools; software process improvement, model-driven engineering, application integration technologies, software quality management, software change and configuration management, geographically distributed software development environments, formal methods, component-based software engineering and commercial-off-the-shelf (COTS) systems, software and systems development methodologies, etc.

 широко July 12-14 GNU Tools Cauldron 2013. Mountain View, California, USA. Topics include: gathering of GNU tools developers, to discuss current/future work, coordinate efforts, exchange reports on ongoing efforts, discuss development plans for the next 12 months, developer tutorials and any other related discussions.

 широко July 16-18 11th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA’2013). Melbourne, Australia. Topics include: parallel and distributed algorithms, and applications; high-performance scientific and engineering computing; middleware and tools; reliability, fault tolerance, and security; parallel/distributed system architectures; tools/environments for parallel/distributed software development; novel parallel programming paradigms; compilers for parallel computers; distributed systems and applications; etc.

 широко July 17-19 18th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS’2013). Singapore. Topics include: verification and validation, security of complex systems, model-driven development, reverse engineering and refactoring, design by contract, agile methods, safety-critical & fault-tolerant architectures, real-time and embedded systems, tools and tool integration, industrial case studies, etc.

 широко July 23-25 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA’2013). Montreal, Canada. Topics include: parallel and distributed algorithms; multi-core architectures; compilers and tools for concurrent programming; synergy of parallelism in algorithms, programming, and architecture; etc.

 широко July 29-30 13th International Conference on Quality Software (QSIC’2013). Nanjing, China. Theme: “Quality of Evolving Software”. Topics include: dynamic analysis, software quality, inspection, fault localization, code review, formal methods, static analysis, proof-based systems, verification techniques combining proofs and tests, testing in multi-core environments, etc. Deadline for submissions: April 5 2013 (technical papers, industry papers).
August 19-23 9th Joint European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'2013). Saint Petersburg, Russia. Topics include: components, development environments and tools, distributed software, embedded and real-time software, maintenance and evolution, model-driven software engineering, parallel and concurrent software, reverse engineering, software architecture, validation, verification, and testing, etc.

Aug 19-20 International Conference on Multicore Software Engineering, Performance, and Tools (MUSEPAT'2013). Topics include: software engineering for multicore systems; specification, modeling and design; programming models, languages, compiler techniques and development tools; verification, testing, analysis; debugging, performance tuning, and security testing; software maintenance and evolution; multicore software issues in scientific computing, embedded and mobile systems; energy-efficient computing; experience reports.

August 19-21 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'2013). Taipei, Taiwan. Topics include: embedded system design practices, software and compiler issues for heterogeneous multi-core embedded platform, real-time scheduling, timing analysis, programming languages and run-time systems, middleware systems, design and analysis tools, case studies and applications, etc.

August 26-30 19th International European Conference on Parallel and Distributed Computing (Euro-Par'2013). Aachen, Germany. Topics include: all aspects of parallel and distributed computing, such as support tools and environments, scheduling, high-performance compilers, distributed systems and algorithms, parallel and distributed programming, multicore and manycore programming, theory and algorithms for parallel computation, etc.


September 08-11 10th International Conference on Parallel Processing and Applied Mathematics (PPAM’2013). Warsaw, Poland. Topics include: multi-core and many-core parallel computing; parallel/distributed algorithms: numerical and non-numerical; scheduling, mapping, load balancing; parallel/distributed programming; tools and environments for parallel/distributed computing; security and dependability in parallel/distributed environments; applications of parallel/distributed computing; etc. Event also includes: workshop on Language-Based Parallel Programming Models. Deadline for submissions: April 21, 2013 (papers).

Sep 10-13 International Conference on Parallel Computing 2013 (ParCo’2013). München, Germany. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments, in particular Parallel programming languages, compilers, and environments; Tools and techniques for generating reliable and efficient parallel code; Best practices of parallel computing on multicore, manycore, and stream processors; etc. Deadline for submissions: July 31, 2013 (full papers).

September 22-28 29th IEEE International Conference on Software Maintenance (ICSM’2013). Eindhoven, the Netherlands. Topics include: software repository analysis and mining; reverse engineering, re-engineering and migration; software refactoring, restructuring and renovation; software and system comprehension; maintenance-related testing; maintenance and evolution processes; software quality improvement; etc. Deadline for submissions: April 17, 2013 (abstracts research track), April 24, 2013 (full papers research track), June 17, 2013 (abstracts), June 24, 2013 (full papers).

Sep 23-24 18th International Workshop on Formal Methods for Industrial Critical Systems (FMICS’2013). Madrid, Spain. Topics include: design, specification, code generation and testing based on formal methods; methods, techniques and tools to support automated analysis, certification, debugging, learning, optimization and transformation of complex, distributed, real-time systems and embedded systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); tools for the development of formal design descriptions; case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardization and industrial forums. Deadline for submissions: May 3, 2013 (papers).
September 23-27 11th **International Conference on Software Engineering and Formal Methods** (SEFM'2013). Madrid, Spain. Topics include: programming languages, program analysis and type theory; formal methods for real-time, hybrid and embedded systems; formal methods for safety-critical, fault-tolerant and secure systems; light-weight and scalable formal methods; tool integration; applications of formal methods, industrial case studies and technology transfer; education and formal methods; etc.

Sep 29 - Oct 04 CBSoft2013 - 17th **Brazilian Symposium on Programming Languages** (SBLP'2013). Brasilia, Distrito Federal, Brazil. Topics include: the fundamental principles and innovations in the design and implementation of programming languages and systems; programming paradigms and styles, including object-oriented, real-time, multithreaded, parallel, and distributed programming; program analysis and verification, including type systems, static analysis and abstract interpretation; programming language design and implementation, including new programming models, programming language environments, compilation and interpretation techniques; etc. Deadline for submissions: April 19, 2013 (abstracts), April 26, 2013 (full papers).

Sep 30 - Oct 03 32nd **International Symposium on Reliable Distributed Systems** (SRDS'2013). Braga, Portugal. Topics include: distributed objects and middleware systems, enabling technologies for dependable applications, formal methods and foundations for dependable distributed computing, analytical or experimental evaluations of dependable distributed systems, secure and trusted systems, high-assurance and safety-critical system design and evaluation, etc. Deadline for submissions: April 2, 2013 (abstracts), April 8, 2013 (full papers).

Oct 03 **ICPP2013 - International Workshop on Embedded Multicore Systems** (EMS'2013). Lyon, France. Topics include: programming models for embedded multicore systems; software for Multicore, GPU, and embedded architectures; real-time system designs for embedded multicore environments; applications for automobile electronics of multicore designs; compiler for worst-case execution time analysis; formal method for embedded systems; etc. Deadline for submissions: May 1, 2013.

October 10-11 **7th International Symposium on Empirical Software Engineering and Measurement** (ESEM'2013). Baltimore, Maryland, USA. Topics include: qualitative methods; replication of empirical studies; empirical studies of software processes and products; industrial experience and case studies; evaluation and comparison of techniques and models; reports on the benefits / costs associated with using certain technologies; empirically-based decision making; quality measurement and assurance; software project experience and knowledge management; etc. Deadline for submissions: June 11, 2013 (short papers, posters).

October 26-28 6th **International Conference on Software Language Engineering** (SLE'2013). Indiana, Indianapolis, USA. Topics include: formalisms used in designing and specifying languages and tools that analyze such language descriptions; language implementation techniques; program and model transformation tools; language evolution; approaches to elicitation, specification, or verification of requirements for software languages; language development frameworks, methodologies, techniques, best practices, and tools for the broader language lifecycle; design challenges in SLE; applications of languages including innovative domain-specific languages or "little" languages; etc. Deadline for submissions: June 7, 2013 (abstracts), June 14, 2013 (full papers).


Oct 26 - 31 **128th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2013). Topics include: any aspect of programming, systems, languages, and applications; any aspect of software development, including requirements, modeling, prototyping, design, implementation, generation, analysis, verification, testing, evaluation, maintenance, reuse, replacement, and retirement of
software systems; large-scale software repositories; tools (such as new languages, program analyses, or runtime systems) or techniques (such as new methodologies, design processes, code organization approaches, and management techniques) that go beyond objects in interesting ways; etc.

Oct 29 - Nov 01  15th **International Conference on Formal Engineering Methods** (ICFEM'2013). Queenstown, New Zealand. Topics include: abstraction and refinement; program analysis; software verification; formal methods for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc. Deadline for submissions: April 1, 2013 (workshops, tutorials), April 15, 2013 (abstracts), April 22, 2013 (full papers).


December 02-05  20th **Asia-Pacific Software Engineering Conference** (APSEC'2013). Bangkok, Thailand. Topics include: software engineering methodologies; software analysis and understanding; software testing, verification and validation; software maintenance and evolution; software quality and measurement; software process and standards; software security, reliability and privacy; software engineering environments and tools; software engineering education; distributed and parallel software systems; embedded and real-time software systems; formal methods in software engineering; etc. Deadline for submissions: 11 June 2013 (workshops), 18 June 2013 (papers), 30 July 2013 (industry track papers, postgraduate symposium papers, tutorials).

December 10  Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

**2014**

A benchmark for safety, security and reliability
– the new updated Ada 2012 language

In an era when software impacts on almost every part of our lives, the need for software that is both safe and secure has never been greater or more urgent. With decades of success in industrial sectors including avionics, aerospace and defence, Ada is generating new interest in domains such as automotive, medical and financial systems, where the cost of software errors can range from lost livelihoods to lost lives. Ada 2012 is the next generation of the world’s premier programming language for engineering safe, secure and reliable software.

The most important enhancements in Ada 2012 are directly related to program “correctness”, namely the introduction of more powerful assertion mechanisms (predicates) into the language: pre- and post-conditions for subprograms, type invariants, and subtype predicates. These are mechanisms that encourage the programmer to better specify the intent of the code they write, and allow the run-time system to verify that this intent is actually achieved. These “programming by contract” features are also beneficial when utilising static analysis tools.

The one-day Ada Conference UK will feature a half-day tutorial on the new capabilities introduced in this latest version. Presented in the morning by Tucker S. Taft, language expert and member of the Ada Rapporteur Group (ARG), this tutorial will introduce these new features to programmers familiar with either Ada or other systems programming languages such as C, C++ or Java. The afternoon session will offer talks from industry and technical experts as well as a selection of vendor talks from the Ada ecosystem.

Registration details and further information on this key event in the Ada calendar can be found at www.ada-uk-conference.co.uk.
## Provisional PROGRAMME

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>08.30</td>
<td>Coffee and Registration</td>
</tr>
<tr>
<td>09.00</td>
<td><strong>Introduction and Welcome:</strong> John Barnes, Event Chairman</td>
</tr>
<tr>
<td>09.00</td>
<td><strong>Ada 2012 Tutorial:</strong> Tucker Taft, SofCheck, Inc.</td>
</tr>
<tr>
<td></td>
<td>(mid-morning break 10.15 to 10.45)</td>
</tr>
<tr>
<td>12.00</td>
<td>Lunch</td>
</tr>
<tr>
<td>13.15</td>
<td><strong>Keynote Speaker:</strong> John C. Knight, University of Virginia, Charlottesville</td>
</tr>
<tr>
<td></td>
<td>“Ada Types - Are They Sufficient?”</td>
</tr>
<tr>
<td>14.00</td>
<td><strong>Technical Track</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Vendor Track</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Speakers</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Definitions</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Proposed Revision to Def Stan 00-56</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Vendor Companies</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Atego</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Altran</strong></td>
</tr>
<tr>
<td></td>
<td><strong>AdaCore</strong></td>
</tr>
<tr>
<td>15.00</td>
<td>Break</td>
</tr>
<tr>
<td>15.30</td>
<td><strong>Technical Track</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Vendor Track</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Definitions</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Mark Richardson, LDRA</strong></td>
</tr>
<tr>
<td></td>
<td><strong>From the model to the target to certification</strong></td>
</tr>
<tr>
<td></td>
<td><strong>IBM</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Wind River</strong></td>
</tr>
<tr>
<td></td>
<td><strong>VectorCast</strong></td>
</tr>
<tr>
<td>16.30</td>
<td><strong>Closing Keynote Speaker:</strong> Robert Dewar, New York University and AdaCore</td>
</tr>
<tr>
<td></td>
<td>“I'm as Mad as Hell, and I'm Not Going To Take This Anymore!”</td>
</tr>
<tr>
<td>17.15</td>
<td>Close</td>
</tr>
</tbody>
</table>

The event website at [www.ada-uk-conference.co.uk](http://www.ada-uk-conference.co.uk) will provide further details as they become available.
The 18th International Conference on Reliable Software Technologies - Ada-Europe 2013 will offer an outstanding technical program, invited talks, an exhibition from Tuesday to Thursday, and a series of tutorials on Monday and Friday.

**Keynotes**


**Giorgio C. Buttazzo**, Scuola Superiore Sant'Anna of Pisa, Italy: Research Challenges in Exploiting Multi-Core Platforms for Real-Time Applications.

**Schedule**

<table>
<thead>
<tr>
<th>Date / Time</th>
<th>Morning</th>
<th>Afternoon</th>
<th>Late afternoon / Evening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday, June 10</td>
<td>Tutorial 1 – 3</td>
<td>Tutorial 4 - 6</td>
<td>Get Together</td>
</tr>
<tr>
<td>Tuesday, June 11</td>
<td>Registration, Keynote 1, Session Multicore and Distributed Systems</td>
<td>Session Industrial Experience 1, Special Session: Ada 2012, Session Ada and Spark Products (Vendor Session)</td>
<td>Ada Europe General Assembly, Welcome Party</td>
</tr>
<tr>
<td>Wednesday, June 12</td>
<td>Keynote 2, Session Dependability</td>
<td>Products (Vendor Session), Session Industrial Experience 2</td>
<td>Conference Dinner at the Botanical Garden</td>
</tr>
<tr>
<td>Thursday, June 13</td>
<td>Keynote 3, Panel: How to Use the Heap in Real-Time Systems</td>
<td>Session Real-Time Systems, Closing Session</td>
<td></td>
</tr>
<tr>
<td>Friday, June 14</td>
<td>Tutorial 7 -9</td>
<td>Tutorial 7, 10-11</td>
<td></td>
</tr>
</tbody>
</table>

For the detailed program including working group meetings, tutorials, social events, exhibition and registration information see www.ada-europe.org/conference2013. For exhibition and sponsorship information please ask at exhibition / sponsor@ada-europe2013.org.

Working group, tutorial or conference registration as well as accommodation requests can only be made online. The early registration is possible until April, 30th and online registration is possible until June, 2nd.

A block of rooms at reduced prices has been reserved at the hotel until April, 30th. Attendees are asked to stay at the conference hotel and make hotel reservations only on the conference registration page.
## Tutorials

### Monday, June 10, 2013

|------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|

### Friday, June 14, 2013

|------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|

### Conference Registration

<table>
<thead>
<tr>
<th>Conference</th>
<th>Early registration by April, 30th</th>
<th>Late/on-site registration after April, 30th</th>
<th>Day registration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member of Ada-Europe, ACM SIG (Ada, BED, PLAN), Ada Germany</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Academic</td>
<td>580 €</td>
<td>730 €</td>
<td>365 €</td>
</tr>
<tr>
<td>Non academic</td>
<td>640 €</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-member</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Academic</td>
<td>640 €</td>
<td>790 €</td>
<td>395 €</td>
</tr>
<tr>
<td>Non academic</td>
<td>700 €</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tutorial Registration

<table>
<thead>
<tr>
<th>Tutorial</th>
<th>Early registration by April, 30th</th>
<th>Late/on-site registration after April, 30th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half day</td>
<td>130 €</td>
<td>145 €</td>
</tr>
<tr>
<td>Full day or two half days on the same day</td>
<td>260 €</td>
<td>290 €</td>
</tr>
</tbody>
</table>

### Accommodation

Please book your accommodation at the conference hotel by using the online conference registration form only.

<table>
<thead>
<tr>
<th>Hotel Seminaris</th>
<th>Double room (single use)</th>
<th>Double room (two persons)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost per night</td>
<td>99 €</td>
<td>134 €</td>
</tr>
<tr>
<td>Weekend rate (Fr-Sun)</td>
<td>86 €</td>
<td>106 €</td>
</tr>
</tbody>
</table>

www.ada-europe.org/conference2013
ACM SIGAda Annual International Conference

High Integrity Language Technology
HILT 2013

Call for Technical Contributions

Developing and Certifying Critical Software

Pittsburgh, Pennsylvania, USA
Fall of 2013 [Mid October to Mid November]

Sponsored by ACM SIGAda
www.sigada.org/conf/hilt2013

SUMMARY

High integrity software must not only meet correctness and performance criteria but also satisfy stringent safety and/or security demands, typically entailing certification against a relevant standard. A significant factor affecting whether and how such requirements are met is the chosen language technology and its supporting tools: not just the programming language(s) but also languages for expressing specifications, program properties, domain models, and other attributes of the software or overall system.

HILT 2013 will provide a forum for experts from academia/research, industry, and government to present the latest findings in designing, implementing, and using language technology for high integrity software. To this end we are soliciting technical papers, experience reports (including experience in teaching), and tutorial proposals on a broad range of relevant topics.

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- New developments in formal methods
- Multicore and high integrity systems
- Object-Oriented Programming in high integrity systems
- High-integrity languages (e.g., SPARK)
- Use of high reliability profiles such as Ravenscar
- Use of language subsets (e.g., MISRA C, MISRA C++)
- Software safety standards (e.g., DO-178B and DO-178C)
- Typed/Proof-Carrying Intermediate Languages
- Contract-based programming (e.g., Ada 2012)
- Model-based development for critical systems
- Specification languages (e.g., Z)
- Annotation languages (e.g., JML)
- Teaching high integrity development
- Case studies of high integrity systems
- Real-time networking/quality of service guarantees
- Analysis, testing, and validation
- Static and dynamic analysis of code
- System Architecture and Design including Service-Oriented Architecture and Agile Development
- Information Assurance
- Security and the Common Criteria / Common Evaluation Methodology
- Architecture design languages (e.g., AADL)
- Fault tolerance and recovery

KINDS OF TECHNICAL CONTRIBUTIONS

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM Ada Letters. The Proceedings will be entered into the widely consulted ACM Digital Library accessible online to university campuses, ACM’s 100,000 members, and the software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, a full paper is required and will appear in the proceedings. Extended abstracts will be double-blind refereed. In 5 pages or less, clearly state the work’s contribution, its relationship with previous work by you and others (with bibliographic references), results to date, and future directions.
EXPERIENCE REPORTS present timely results and “lessons learned”. Submit a 1-2 page description of the project and the key points of interest. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather groups of experts on particular topics. Panelists present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation and, if selected, a subsequent abstract for a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for ACM Ada Letters.

WORKSHOPS are focused sessions that allow knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. Workshop proposals, up to 5 pages in length, will be selected based on their applicability to the conference and potential for attracting participants.

TUTORIALS can address a broad spectrum of topics relevant to the conference theme. Submissions will be evaluated based on applicability, suitability for presentation in tutorial format, and presenter’s expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half day or full day).

HOW TO SUBMIT: Send in Word, PDF, or text format:

<table>
<thead>
<tr>
<th>Submission</th>
<th>Deadline</th>
<th>Send to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals</td>
<td>June 29, 2013</td>
<td>Tucker Taft, Program Chair <a href="mailto:taft@adacore.com">taft@adacore.com</a></td>
</tr>
<tr>
<td>Industrial presentation proposals</td>
<td>August 1, 2013 (overview) September 30, 2013 (abstract)</td>
<td>John McCormick, Tutorials Chair <a href="mailto:mccormick@cs.uni.edu">mccormick@cs.uni.edu</a></td>
</tr>
<tr>
<td>Tutorial proposals</td>
<td>June 29, 2013</td>
<td></td>
</tr>
</tbody>
</table>

At least one author is required to register and make a presentation at the conference.

FURTHER INFORMATION

CONFERENCE GRANTS FOR EDUCATORS: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for 2 days of conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact Prof. Michael B. Feldman (MFeldman@gwu.edu)

OUTSTANDING STUDENT PAPER AWARD: An award will be given to the student author(s) of the paper selected by the program committee as the outstanding student contribution to the conference.

SPONSORS AND EXHIBITORS: Please contact Greg Gicca (gicca@adacore.com) to learn the benefits of becoming a sponsor and/or exhibitor at HILT 2013.

IMPORTANT INFORMATION FOR NON-US SUBMITTERS: International registrants should be particularly aware and careful about visa requirements, and should plan travel well in advance. Visit the conference website for detailed information pertaining to visas.

ANY QUESTIONS?

Please send email to SIGAda.HILT2013@acm.org or contact the Conference Chair (Jeff Boleng, jjboleng@SEI.CMU.EDU), Program Chair (Tucker Taft, taft@adacore.com), SIGAda’s Vice-Chair for Meetings and Conferences (Alok Srivastava, alok.srivastava@tasc.com), or SIGAda’s Chair (Ricky E. Sward, rsward@mitre.org).
FOR IMMEDIATE RELEASE

Ada 2012 Language Standard Approved by ISO
Language revision adds contract-based programming, multicore support, and other advanced features

GENEVA, Switzerland, December 18, 2012 – The Ada Resource Association (ARA) and Ada-Europe today announced the approval and publication of the latest version of the Ada programming language by the Geneva-based International Organization for Standardization (ISO). The language revision, known as Ada 2012, was under the auspices of ISO/IEC JTC1/SC22/WG9 and was conducted by the Ada Rapporteur Group (ARG) subunit of WG9, with sponsorship in part from the ARA and Ada-Europe. The formal approval of the standard was issued on November 20 by ISO/IEC JTC 1, and the standard was published on December 15.

Ada 2012 brings significant enhancements to Ada, most notably in the area of “contract-based programming.” New features here include the ability to specify preconditions and postconditions for subprograms, and invariants for private (encapsulated) types. These take the form of Boolean expressions that can be interpreted (under programmer control) as run-time conditions to be checked. The contract-based programming features fit in smoothly with Ada’s Object-Oriented Programming model, and support the type substitutability guidance supplied in the Object-Oriented Technologies and Related Techniques Supplement (DO-332) to the new avionics software safety standard DO-178C / ED-12C.

Other new features in Ada 2012 include enhancements to the containers library, additional expressiveness through features such as conditional expressions and more powerful iterators, and support for multicore platforms (task affinities, and the extension of the Ravenscar profile – standardized in Ada 2005 as an efficient and predictable tasking subset for high-integrity real-time systems – to multiprocessor and multicore environments).

A technical summary of Ada 2012, together with an explanation of the language’s benefits and a set of links to further information, is available at www.ada2012.org, a website maintained by the Ada Resource Association.

“Ada 2012 is a major advance in the state of the art in programming languages,” said Dr. Edmond Schonberg, Rapporteur of the ARG. “The new features answer real user needs, and help cement Ada’s reputation as a language of choice for systems where reliability, safety, and security are paramount.”
“I would like to give special thanks to Randy Brukardt for his editorial work on the Language Reference Manual, to Ed Schonberg and the Ada Rapporteur Group (ARG) for their excellent work in developing the language revision, and to all members of WG 9 in producing a quality document that achieved unanimous approval from our parent organization,” said Dr. Joyce Tokar, Convenor of WG9. “Ada 2012 is a significant technical accomplishment.”

With the growing complexity of software systems in most aspects of our daily professional and personal life, program correctness is a paramount concern. Ada 2012 provides outstanding solutions to that end, which can be applied both in industry for production software development, and in academia for teaching and research.

About the Ada Resource Association
The Ada Resource Association (ARA) is a non-profit organization chartered to support the continued evolution of the Ada language and its infrastructure, to serve as a source of information about Ada and its usage, and to promote Ada as a language for effective software engineering. To these ends the ARA maintains the Ada Information Clearinghouse website www.adaic.org and has provided funding for the development and maintenance of the Ada language standard and the Ada Conformance Assessment Test Suite. For information about the ARA, including sponsorship opportunities, please visit www.adaresource.com. The ARA is headquartered in Oakton, VA (US).

About Ada-Europe
Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry in Europe. Its flagship event is the annual international conference on reliable software technologies, a high-quality technical and scientific event that has been successfully running in the current format for the last 17 years. Ada-Europe has member organizations all over the continent, in Belgium, Denmark, France, Germany, Spain, Sweden, and Switzerland, as well as individual members in many other countries. For information about Ada-Europe, its charter, activities and sponsors, please visit: www.ada-europe.org. Ada-Europe is headquartered in Brussels, Belgium.

Organization Contacts

**Ada Resource Association**
Ben Bros gol, ARA President
bros gol@adacore.com

**Ada-Europe**
Tullio Vardanega, Ada-Europe President
pres ident@ada-europe.org

Press Contacts

**Ada Resource Association**
Jessie Glockner
Rainier Communications
Tel: +1-508-475-0025 x140
jglockner@rainierco.com
http://twitter.com/JessieGlockner

**Ada-Europe**
Dirk Craeynest, Ada-Europe Vice-president
c/o KU Leuven
Department of Computer Science
dirk.craeynest@cs.kuleuven.be
Rationale for Ada 2012: 6 Predefined library

John Barnes
John Barnes Informatics, 11 Albert Road, Caversham, Reading RG4 7AN, UK; Tel: +44 118 947 4125; email: jgpb@jbinfo.demon.co.uk

Abstract
This paper describes various relatively minor improvements to the predefined library in Ada 2012. The major changes concerning the container library will be described in a later paper.

Keywords: rationale, Ada 2012

1 Overview of changes

The WG9 guidance document [1] does not specifically identify problems in this area other than through a general exhortation to remedy shortcomings.

We have already discussed the additional library packages in the area of tasking and real-time in a previous paper. There are also many additional library packages concerning containers and these will be discussed in a later paper. The following Ada issues cover the relevant changes in other areas and are described in detail in this paper:

1 Bounded containers and other container issues
31 Add a From parameter to Find_Token
49 Extend file name processing in Ada.Directories
127 Adding locale capabilities
137 String encoding package
185 Wide_Character and Wide_Wide_Character classification and folding
233 Questions on locales
266 Use latest version of ISO/IEC 10646
283 Stream_IO should be preelaborated
285 Defaulted environment variable queries
286 Internationalization of Ada

These changes can be grouped as follows.

A number of enhancements concern strings and characters. These include comprehensive new packages to support conversions between strings (and wide strings and wide-wide strings) and the UTF-8 and UTF-16 encodings (137). It is important to note that Ada 2012 directly supports source code in UTF-8 (286). Additional facilities are also provided for the classification of characters and new packages added for similar operations on wide characters and wide wide characters (185, 266). A minor change is the provision of a further procedure Find_Token with an additional parameter giving the start of the search (31).

The file name processing in Ada.Directories is enhanced to overcome some shortcomings (49).

A new package is added to enable a program to identify the locale in which it is being used (127, 233).

There are a number of additional facilities regarding hashing and case insensitive comparisons. The hashing issues really relate to containers but are briefly mentioned here for completeness (1, 286).

Finally, other improvements are that the package Ada.Streams.Stream_IO is now preelaborated (283) and that an additional function Value is added to the package Ada.Environment_Variables (285).

2 Strings and characters

Ada 95 added a number of packages for manipulating strings and characters. Three child packages of Ada.Strings enable the manipulation of fixed length, bounded and unbounded strings. They are Ada.Strings.Fixed, Ada.Strings.Bounded and Ada.Strings.Unbounded. The packages have many subprograms with similar facilities.

In particular there are functions Index and Index_Non_Blank which search through a string and return the index of the first character satisfying some criteria and procedures Find_Token which search through a string and find the first instance of a slice satisfying some other criteria.

As originally defined in Ada 95 these subprograms all started the search at the beginning of the string. This proved to be somewhat inconvenient and so in Ada 2005, versions of the functions Index and Index_Non_Blank with an extra parameter From were added to enable the search to be started at any position. However, the fact that versions of the procedures Find_Token with an extra parameter From should also have been added was overlooked. This is remedied in Ada 2012.

So in Ada 2012 corresponding additional subprograms Find_Token are added to the appropriate packages. They are

```
procedure Find_Token(Source: in String;
                          Set: in Maps.Character_Set;
                          From: in Positive;
                          Test: in Membership;
                          First: out Positive;
                          Last: out Natural);

procedure Find_Token(Source: in Bounded_String;
                          Set: in Maps.Character_Set;
```
Note also that the wording for Find_Token is modified to make it clear that the values of First and Last identify the longest possible slice starting at From. If no characters satisfy the criteria then First is set to From and Last is set to zero.

The existing procedures Find_Token are now defined as calls of the new ones with From set to Source'First.

The encodings UTF-8 and UTF-16 are now widely used but Ada 2005 provides no mechanisms to convert between these encodings and the types String, Wide_String, and Wide_Wide_String.

The encoding UTF-8 works in terms of raw bytes and is straightforward; it is defined in Annex D of ISO/IEC 10646. However, UTF-16 comes in two forms according to whether the arrangement of two bytes into a 16-bit word uses big-endian or little-endian packing. So there are two forms UTF-16BE and UTF-16LE; they are defined in Annex C of ISO/IEC 10646.

The different encodings can be distinguished by a special value known as a BOM (Byte Order Mark) at the start of the string. So we have BOM_8, BOM_16BE, BOM_16LE, and just BOM_16 (for wide strings).

To support these encodings, Ada 2012 includes the following five new packages

- Ada.Strings.UTF_Encoding
- Ada.Strings.UTF_Encoding.Conversions
- Ada.Strings.UTF_Encoding.Wide_Wide_Strings
- Ada.Strings.UTF_Encoding.Wide_Strings
- Ada.Strings.UTF_Encoding.Strings

The first package declares items that are used by the other packages. It is

```ada
package Ada.Strings.UTF_Encoding is
  pragma Pure(UTF_Encoding);
  type Encoding_Scheme is (UTF_8, UTF_16BE, UTF_16LE);
  subtype UTF_String is String;
  subtype UTF_8_String is String;
  subtype UTF_16_Wide_String is Wide_String;
  Encoding_Error: exception;

  BOM_8: constant UTF_8_String :=
    Character'Val(16#EF#) &
    Character'Val(16#BF#);
  BOM_16BE: constant UTF_String :=
    Character'Val(16#FE#) &
    Character'Val(16#FF#);
  BOM_16LE: constant UTF_String :=
    Character'Val(16#FF#) &
    Character'Val(16#FE#);
  BOM_16: constant UTF_16_Wide_String :=
    (1 => Wide_Character'Val(16#FEFF#);

  function Encoding(Item: UTF_String;
      Default: Encoding_Scheme := UTF_8) return Encoding_Scheme;

end Ada.Strings.UTF_Encoding;
```

Note that the encoded forms are actually still held in objects of type String or Wide_String. However, in order to aid understanding, the subtypes UTF_String, UTF_8_String and UTF_16_Wide_String are introduced and these should be used when referring to objects holding the encoded forms.

The type Encoding_Scheme defines the various schemes. Note that an encoded string might or might not start with the identifying BOM; it is optional. The function Encoding takes a UTF_String (that is a plain old string), checks the BOM if present and returns the value of Encoding_Scheme identifying the scheme. If there is no BOM then it returns the value of the parameter Default which itself by default is UTF_8.

Note carefully that the function Encoding does not do any encoding — that is done by functions Encode in the other packages which will be described in a moment. Note also that there is no corresponding function Encoding for wide strings; that is because there is only one relevant scheme corresponding to UTF_16_Wide_String, namely that with BOM_16.

We will now look at the other packages. The package UTF_Encoding.Strings contains functions Encode and Decode which convert between the raw type String and the UTF forms. Similar packages apply to wide and wide wide strings. The package UTF_Encoding.Conversions contains functions Convert which convert between the various UTF forms.

The package for the type String is

```ada
package Ada.Strings.UTF_Encoding.Strings is
  pragma Pure(Strings);

  function Encode(Item: String;
      Output_Scheme: Encoding_Scheme := Encoding_Scheme;
      Output_BOM: Boolean := False) return UTF_String;

  function Encode(Item: String;
      Output_BOM: Boolean := False) return UTF_8_String;

  function Encode(Item: String;
      Output_BOM: Boolean := False) return UTF_16_Wide_String;
```

The package for the type Wide_String

```ada
package Ada.Strings.UTF_Encoding.Wide_Strings is
  pragma Pure(Wide_Strings);

  function Encode(Item: Wide_String;
      Output_Scheme: Encoding_Scheme := Encoding_Scheme;
      Output_BOM: Boolean := False) return UTF_String;

  function Encode(Item: Wide_String;
      Output_BOM: Boolean := False) return UTF_8_String;

  function Encode(Item: Wide_String;
      Output_BOM: Boolean := False) return UTF_16_Wide_String;
```

The package for the type Wide_Wide_String

```ada
package Ada.Strings.UTF_Encoding.Wide_Wide_Strings is
  pragma Pure(Wide_Wide_Strings);

  function Encode(Item: Wide_Wide_String;
      Output_Scheme: Encoding_Scheme := Encoding_Scheme;
      Output_BOM: Boolean := False) return UTF_String;

  function Encode(Item: Wide_Wide_String;
      Output_BOM: Boolean := False) return UTF_8_String;

  function Encode(Item: Wide_Wide_String;
      Output_BOM: Boolean := False) return UTF_16_Wide_String;
```

The package for the type UTF_String

```ada
package Ada.Strings.UTF_Encoding.Conversions is
  pragma Pure(Conversions);

  function Convert(Item: UTF_String;
      Default: Encoding_Scheme := UTF_8) return Wide_String;
```

The package for the type UTF_8_String

```ada
package Ada.Strings.UTF_Encoding.Wide_Wide_Conversions is
  pragma Pure(Wide_Wide_Conversions);

  function Convert(Item: UTF_8_String;
      Default: Encoding_Scheme := UTF_8) return Wide_Wide_String;
```

The package for the type UTF_16_Wide_String

```ada
package Ada.Strings.UTF_Encoding.Wide_Strings is
  pragma Pure(Wide_Strings);

  function Convert(Item: UTF_16_Wide_String;
      Default: Encoding_Scheme := UTF_16_Wide) return Wide_String;
```

The package for the type UTF_16_Wide_String

```ada
package Ada.Strings.UTF_Encoding.Conversions is
  pragma Pure(Conversions);

  function Convert(Item: UTF_16_Wide_String;
      Default: Encoding_Scheme := UTF_16_Wide) return Wide_Wide_String;
```

Note also that the wording for Find_Token is modified to make it clear that the values of First and Last identify the longest possible slice starting at From. If no characters satisfy the criteria then First is set to From and Last is set to zero.

The existing procedures Find_Token are now defined as calls of the new ones with From set to Source'First.

The encodings UTF-8 and UTF-16 are now widely used but Ada 2005 provides no mechanisms to convert between these encodings and the types String, Wide_String, and Wide_Wide_String.

The encoding UTF-8 works in terms of raw bytes and is straightforward; it is defined in Annex D of ISO/IEC 10646. However, UTF-16 comes in two forms according to whether the arrangement of two bytes into a 16-bit word uses big-endian or little-endian packing. So there are two forms UTF-16BE and UTF-16LE; they are defined in Annex C of ISO/IEC 10646.

The different encodings can be distinguished by a special value known as a BOM (Byte Order Mark) at the start of the string. So we have BOM_8, BOM_16BE, BOM_16LE, and just BOM_16 (for wide strings).

To support these encodings, Ada 2012 includes the following five new packages

- Ada.Strings.UTF_Encoding
- Ada.Strings.UTF_Encoding.Conversions
- Ada.Strings.UTF_Encoding.Wide_Wide_Strings
- Ada.Strings.UTF_Encoding.Wide_Strings
- Ada.Strings.UTF_Encoding.Strings

The first package declares items that are used by the other packages. It is
function Decode(Item: UTF_String;
   Input_Scheme: Encoding_Scheme)
   return String;

function Decode(Item: UTF_8_String;)
   return String;

function Decode(Item: UTF_16_Wide_String;)
   return String;

end Ada.Strings.UTF_Encoding.Strings;

The functions Encode take a string and return it encoded. The first function has a parameter Output_Scheme which determines whether the encoding is to be to UTF_8, UTF_16BE or UTF_16LE. The second function is provided as a convenience for the common case of encoding to UTF_8 and the third function is necessary for encoding to UTF_16_Wide_String. In all cases there is a final optional parameter indicating whether or not an appropriate BOM is to be placed at the start of the encoded string.

The functions Decode do the reverse. Thus the first function takes a value of subtype UTF_String and a parameter Input_Scheme giving the scheme to be used and returns the decoded string. If a BOM is present which does not match the Input_Scheme, then the exception Encoding_Error is raised. The second function is a convenience for the common case of decoding from UTF_8 and the third function is necessary for decoding from UTF_16_Wide_String; again, if a BOM is present that does not match the expected scheme then Encoding_Error is raised.

In all cases all the strings returned have a lower bound of 1.

The packages UTF_Encoding.Strings and UTF_Encoding.Wide_Wide_Strings are identical except that the type String is replaced by Wide_String or Wide_Wide_String throughout.

Finally, the package for converting between the various UTF forms is as follows

package Ada.Strings.UTF_Encoding.Conversions is
   pragma Pure(Conversions);

   function Convert(Item: UTF_String;
      Input_Scheme: Encoding_Scheme
      Output_Scheme: Encoding_Scheme;
      Output_BOM: Boolean := False)
      return UTF_String;

   function Convert(Item: UTF_8_String;
      Input_Scheme: Encoding_Scheme;
      Output_BOM: Boolean := False)
      return UTF_16_Wide_String;

   function Convert(Item: UTF_16_Wide_String;
      Input_Scheme: Encoding_Scheme;
      Output_BOM: Boolean := False)
      return UTF_String;

end Ada.Strings.UTF_Encoding.Conversions;

The purpose of these should be obvious. The first converts between encodings held as strings with parameters indicating both the Input_Scheme and the Output_Scheme. If the input string has a BOM that does not match the Input_Scheme then the exception Encoding_Error is raised. The final optional parameter indicates whether or not an appropriate BOM is to be placed at the start of the converted string.

The other functions convert between UTF encodings held as strings and wide strings. Two give the explicit Input_Scheme or Output_Scheme and two are provided for convenience for the common case of UTF_8.

The final topic in this section concerns the classification and folding of characters and strings. The package Ada.Characters.Handling was introduced in Ada 95; this contains various classification functions such as Is_Lower, Is_Digit and so on. This package also contains functions such as To_Upper and To_Lower which convert characters to upper case or lower case; such conversions are often referred to as case folding operations.


It should be noticed that these new packages are children of Ada.Wide_Characters and Ada.Wide_Wide_Characters respectively. These packages were introduced in Ada 2005 but are empty other than for pragmas Pure.

The additional functions in Ada.Characters.Handling are

function Is_Line_Terminator ... ...
function Is_Mark(Item: Character) return Boolean;
function Is_Other ... ...
function Is_Punctuation_Connector ... ...
function Is_Space ... ...

In each case they have a single parameter Item of type Character and return a result of type Boolean.

The meanings are as follows

Is_Line_Terminator — returns True if Item is one of Line_Feed (10), Line_Tabulation (11), Form_FEED (12), Carriage_Return (13), or Next_Line (133).

Is_Mark — always returns False.

Is_Other_Format — returns True if Item is Soft_Hyphen (171).

Is_Punctuation_Connector — returns True if Item is Low_Line (95); this is often known as Underscore.
Readers might feel that Is_Mark is a foolish waste of time. However, it is introduced because the corresponding functions in the new packages for wide and wide wide characters can return True.

An important point is that these classifications enable a compiler to analyze Ada source code without direct reference to the definition of ISO/IEC 10646. Note further that case insensitive text comparison which is useful for the analysis of identifiers is now provided by new functions described in Section 5 below.

The new package Wide_Characters.Handling is very similar to the package Characters.Handling (as modified by the additional functions just described) with Character and String everywhere replaced by Wide_Character and Wide_String. However, there are no functions corresponding to Is_Basic, Is_ISO_646, To_Basic and To_ISO_646. In the case of Is_Basic this is because there is no categorization of Basic in 10646. In the case of ISO-646 it is not really necessary because it would seem rather unlikely that one would want to check a wide character WC to see if it was one of the 7-bit ISO-646 set. In any event, one could always write

```
WC in Wide_Characters'POS(0) .. Wide_Characters'POS(127)
```

The package Wide_Characters.Handling also has the new function Character_Set_Version thus

```
function Character_Set_Version return String;
```

The string returned identifies the version of the character set standard being used. Typically it will include either "10646:" or "Unicode". The reason for introducing this function is because the categorization of some wide characters depends upon the version of 10646 or Unicode being used. So rather than specifying that the package uses a particular set (which might be a nuisance in the future if the character set standard changes), it seemed more appropriate to enable the program to find out exactly which version is being used. For most programs, it won't matter at all of course.

Note that there is no corresponding function in Ada.Characters.Handling. This is because the set used for the type Character is frozen as at 1995 and the classification functions defined for the type Character are frozen as well. It might be that classifications for wide and ever wider characters might change in the future for some obscure characters but the programmer can rest assured that Character is forever reliable.

So Wide_Characters.Handling in essence is

```
package Ada.Wide_Characters.Handling is
  pragma Pure(Handling);
  function Character_Set_Version return String;
  function Is_Control(Item: Wide_Character) return Boolean;
```

The new package Wide_Wide_Characters.Handling is the same as Wide_Characters.Handling with Wide_Character and Wide_String replaced by Wide_Wide_Character and Wide_Wide_String throughout.

### 3 Directories

The package Ada.Directories was introduced in Ada 2005. However, experience with its use has revealed a number of shortcomings which are rectified in Ada 2012.

Three specific problems are mentioned in AI-49.

First, it is not possible to concatenate a root directory such as "/tmp" with a relative pathname such as "public/file.txt" using the procedure Compose thus

```
The_Path: String := Compose("/tmp", "public/file.txt");
```

This is because the second parameter of Compose has to be a simple name such as just "file" if there is no extension parameter. If we supply the extension parameter thus

```
The_Path: String := Compose("/tmp", "public/file", "txt");
```

then the second parameter has to be just a base name such as "public".

Another problem is that there is no sensible way to check for a root directory. Thus suppose the string $S$ is a directory name and we want to see whether it is just a root such as "$/$" in Unix then the only thing that we can do is write

```
Containing_Directory(S)
```

which will raise Use_Error which is somewhat ugly.

We could write $\#S = "/"$ then but this would not be portable from Unix to other systems. Indeed, the whole purpose of providing file name operations in Ada.Directories is so that file names can be manipulated in an abstract manner without fiddling with text strings.

The third problem concerns case sensitivity. At the moment it is not possible to write portable programs because operating systems differ in their approach to this issue.

This last problem is solved by adding an enumeration type Name_Case.Kind and a function Name_Case_Equivalence to the file and directory name operations of the package Ada.Directories. So in outline we now have

```
with Ada.IO_Exceptions; with Ada.Calendar;
package Ada.Directories is
  ... -- File and directory name operations:
  function Full_Name(Name: String) return String;
  function Simple_Name(Name: String) return String;
  function Containing_Directory(Name: String) return String;
end Ada.Directories.
```

function Extension(Name: String) return String;
function Base_Name(Name: String) return String;
function Compose(Containing_Directory: String := "";
Name: String;
Extension: String := ")
return String;

type Name_Case_Kind := (Unknown, Case_Sensitive,
Case_Insensitive, Case_Preserving);

function Name_Case_Equivalence(Name: String)
return Name_Case_Kind;

-- File and directory queries:
-- and so on
end Ada.Directories;

The function Name_Case_Equivalence returns the file
name equivalence rule for the directory containing Name. It
raises Name_Error if Name is not a Full_Name.

It returns Case_Sensitive if file names that differ only in
the case of letters are considered to be different. If file
names that differ only in the case of letters are considered
to be the same, then it returns Case_Preserving if the name
has the case of the file name used when a file is created and
Case_Insensitive otherwise. It returns Unknown if the name
equivalence rule is not known.

We thus see that Unix and Linux are Case_Sensitive,
Windows is Case_Preserving, and historic systems such as
CP/M and early MS/DOS were Case_Insensitive.

The other problems are solved by the introduction of an
optional child package for dealing with systems with
hierarchical file names. Its specification is

package Ada.Directories.Hierarchical_File_Names is

  function Is_Simple_Name(Name: String) return Boolean;
  function Is_Root_Directory_Name(Name: String) return Boolean;
  function Is_Parent_Directory_Name(Name: String) return Boolean;
  function Is_Current_Directory_Name(Name: String) return Boolean;
  function Is_Full_Name(Name: String) return Boolean;
  function Is_Relative_Name(Name: String) return Boolean;

  function Simple_Name(Name: String)
renames Ada.Directories.Simple_Name;

  function Containing_Directory(Name: String)
renames Ada.Directories.Containing_Directory;

  function Initial_Directory(Name: String) return String;
  function Relative_Name(Name: String) return String;

  function Compose(Directory: String := "";
Relative_Name: String;
Extension: String := ")
return String;

end Ada.Directories.Hierarchical_File_Names;

Note that the six functions, Full_Name, Simple_Name,
Containing_Directory, Extension, Base_Name and
Compose in the existing package Ada.Directories just
manipulate strings representing file names and do not in
any way interact with the actual external file system. The
same applies to many of the new functions such as
Is_Simple_Name.

In particular, Is_Root_Directory_Name returns true if the
string is syntactically a root and so cannot be decomposed
further. It therefore solves the second problem mentioned
earlier. Thus

    Is_Root_Directory_Name("/")

returns true for Unix. In the case of Windows "C:" and
"\Computer\Share" are roots.

The function Is_Parent_Directory_Name returns true if and
only if the Name is "." for both Unix and Windows.

The function Is_Current_Directory_Name returns true if and
only if Name is "." for both Unix and Windows.

The function Is_Full_Name returns true if the leftmost part
of Name is a root whereas Is_Relative_Name returns true if
Name allows identification of an external file but is not a
full name. Note that relative names include simple names
as a special case.

The functions Simple_Name and Containing_Directory are
just renamings of those in the parent package and are
provided for convenience.

Finally, the functions Initial_Directory, Relative_Name and
Compose provide the ability to manipulate relative file
names and so solve the problem with Compose mentioned
at the beginning of this section.

Thus Initial_Directory returns the leftmost directory part
of Name and Relative_Name returns the entire full name apart
from the initial directory portion.

If we apply Relative_Name to a string that is just a single
part of a name then Name_Error is raised. In particular this
happens if Relative_Name is applied to a name which is a
Simple Name, a Root Directory Name, a Parent Directory
Name or a Current Directory Name.

The function Compose is much like Compose in the parent
package except that it takes a relative name rather than a
simple name. It therefore allows us to write

    The_Path: String := Compose("/tmp", "public/file.txt");

as required.

The result of calling Compose is a full name if
Is_Full_Name(Directory) is true and otherwise is a relative
name.

4 Locale

When writing portable software it is often necessary to
know the locality in which the software is to be run. Two
key items are the country and the language (human
language that is, not programming language).
To enable this to be done, Ada 2012 includes the following package

```ada
package Ada.Locales is
  pragma Preelaborate(Locales);
  pragma Remote_Types(Locales);
  type Language_Code is array (1 .. 3) of Character
    range 'A' .. 'Z';
  type Country_Code is array (1 .. 2) of Character
    range 'A' .. 'Z';
  Language_Unknown: constant Language_Code := "und";
  Country_Unknown: constant Country_Code := "ZZ";
  function Language return Language_Code;
  function Country return Country_Code;
end Ada.Locales;
```

The various country codes and language codes are defined in ISO/IEC 3166-1:2006 and ISO/IEC 639-3:2007 respectively.

Knowledge of the locale is important for writing programs where the convention for certain information varies. Thus in giving a date we might want to add the name of the day of the week and clearly in order to do this we need to know what language to use. An earlier (really grotesque) attempt at providing this information introduced a host of packages addressing many issues. However, it was decided that for simplicity and indeed reliability all that is really needed is to know the language to use and the country.

Canada is interesting in that it has just one country code ("CA") but two language codes ("eng" and "fra"). In Quebec, a decimal value for a million dollars and one cent is written as $1,000,000.01 whereas in English language parts it is written as $1,000,000.01 with the comma and stop interchanged.

Sometimes, several locales might be available on a target. Some environments define a system locale and a locale for the current user. In the case of an Ada program the active locale is the one associated with the partition of the current task.

5 Hashing and comparison

New library functions are added for case insensitive comparisons and hashing. Thus we have

```ada
function Ada.Strings.Equal_CaseInsensitive (Left, Right: String) return Boolean;
pragma Pure(Ada.Strings.Equal_CaseInsensitive);
```

This simply compares the strings Left and Right for equality but ignoring case. Thus

```
Equal_CaseInsensitive("Pig", "PIG")
```

is true.

The function Ada.Strings.Fixed.Equal_CaseInsensitive is a renaming of the above. There are also similar functions Ada.Strings.Bounded.Equal_CaseInsensitive for bounded and Ada.Strings.Unbounded.Equal_CaseInsensitive for unbounded strings. And, as expected, there are similar functions for wide and wide wide versions.

Note that the comparison for strings can be phrased as convert to lower case and then compare. But this does not always work for wide and wide wide strings. The proper terminology is "locale-independent case folding and then compare".

Although it comes to the same thing for Latin-1 characters there are problems with some character sets where there is not a one-one correspondence between lower case and upper case. This used to apply to English with the two forms of lower case S and still applies to the corresponding letters in Greek where the upper case character is Σ and there are two lower case versions namely σ and ς. So

```
Ada.Wide.Strings.Equal_CaseInsensitive("ΣΩ", "σω")
```

returns true. Note that if we convert to lower case first then it would not be true.

Furthermore there is also

```ada
function Ada.Strings.Less_CaseInsensitive
  (Left, Right: String) return Boolean;
pragma Pure(Ada.Strings.Less_CaseInsensitive);
```

which does a lexicographic comparison.

As expected there are similar functions for fixed, bounded and unbounded strings and, naturally, for wide and wide wide versions.

Ada 2005 has functions for hashing such as

```ada
with Ada.Containers;
function Ada.Strings.Hash(Key: String)
  return Containers.Hash_Type;
```

Ada 2012 adds case insensitive versions as well such as

```ada
with Ada.Containers;
function Ada.Strings.Hash_CaseInsensitive
  (Key: String) return Containers.Hash_Type;
```

There are also fixed, bounded and unbounded versions and the inevitable wide and wide wide ones as well.

6 Miscellanea

The first item is that the package Stream_IO should be marked as preelaborated. So in Ada 2012 it now begins

```ada
with Ada.IO_Exceptions;
package Ada.Streams.Stream_IO is
  pragma Preelaborate(Stream_IO);
...
```

The reason for making this change concerns the use of input–output in preelaborated packages. The normal input–output packages such as Text_IO are not preelaborated and so cannot be used in packages that are themselves preelaborated. This makes preelaborated packages awkward to debug since they cannot do straightforward output for monitoring purposes. To make packages such as Text_IO preelaborated is essentially impossible because
they involve local state. However, no such problem exists with Stream_IO, and so making it preelaborated means that it can be used to implement simple logging facilities in other preelaborated packages.

In principle, there is a similar problem with pure units. But they cannot change state anyway and so cannot do output since that changes the state of the environment. They just have to be written correctly in the first place.

(I have been told that there are naughty ways around this with pure packages but I will not contaminate innocent minds with the details.)

The package Ada.Environment_Variables was introduced in Ada 2005 as follows

```ada
package Ada.Environment_Variables is
   pragma Preelaborate(Ada.Environment_Variables);
   function Value(Name: String) return String;
   function Exists(Name: String) return Boolean;
   procedure Set(Name: in String; Value: in String);
   procedure Clear(Name: in String);
   procedure Clear;
   procedure Iterate(Process: not null access procedure (Name: String; Value: in String));
end Ada.Environment_Variables;
```

If we do not know whether an environment variable exists then we can check by calling Exists prior to accessing the current value. Thus a program might be running in an environment where we might expect an environment variable "Ada" whose value indicates the version of Ada currently supported.

Rationale for Ada 2012: 6 Predefined library

So as in [2] we might write

```ada
if not Exists("Ada") then
   raise Horror;
end if;
```

```ada
Put ("Current Ada is "); Put_Line(Value("Ada"));
```

But this raises a possible race condition. After determining that Ada does exist some malevolent process (such as another Ada task or an external human agent) might execute Clear("Ada"); and then the call of Value("Ada") will raise Constraint_Error.

The other race condition might arise as well. Having decided that Ada does not exist and so taking remedial action some kindly process might have created Ada.

These problems are overcome in Ada 2012 by the introduction of an additional function Value with a default parameter

```ada
function Value(Name: String; Default: String);
```

Calling this version of Value returns the value of the variable if it exists and otherwise returns the value of Default.

References


© 2013 John Barnes Informatics.
Designing and Checking Coding Standards for Ada

Jean-Pierre Rosen
Adalog, 2 rue du Docteur Lombard, 92130 Issy-Les-Moulineaux, France; Tel: +33 1 45 29 21 52; email: rosen@adalog.fr

Abstract
This paper is an extended summary of the tutorial given at Ada-Europe 2012. It presents the challenges of establishing a coding standard, not just for the sake of having one, but with the goal of actually improving the quality of software.

Keywords: Ada, coding standard, checking tools.

Introduction
Most companies have developed coding standards, but few have conducted an analysis of the value, consistency, and efficiency of the standard. This means not only having "good" rules, but also having rules that are understood, accepted, and adhered to by the programming team. The tutorial explored the issues involved in achieving these goals.

1 What is a coding standard?
A coding standard is a document that defines a number of requirements that are to be obeyed during software development. This document often includes coding rules and coding guidelines. Although these are often mixed together in a single document, it is important to understand the differences, since they serve very different purposes.

Coding rules are formally defined, and safe for special (and justified) cases, and should always be followed by developers. They define patterns that are either required or forbidden. They can (and should) be checked by automated tools.

Coding guidelines are informal recommendations, expressing best practices, how-to's, general principles, that are by nature hard or even impossible to enforce by automated tools. Whether a coding guideline has been satisfactorily followed is often a matter of judgement.

For example, "All variables shall be written in title-case" is a programming rule. "All variables shall have readable names that express their purpose" is a programming guideline.

1.1 Purpose of coding standards
A coding standard may serve several different purposes:

- Uniformity: provide a common look-and-feel at project or even company level.
- Error prevention: avoid use of error-prone or dangerous features.
- Maintainability: ease of maintenance and evolution by others than the initial coder.
- Safety and domain specific rules: enforce project specific requirements: validation, efficiency, provability, etc.

But there is another reason for having a coding standard that is unfortunately often encountered: the coding standard is there just because some process (DO-178B/C, EN-50128, and the like) requires one to exist. This can lead to having rules just for the sake of having rules, without a clear understanding of their purpose, with the result that the coding standard is full of unjustified rules, and may end up being an obstacle to software development rather than a factor of software quality.

Providing guidance for avoiding misguided rules and fostering effective ones is the purpose of this tutorial.

1.2 Strength of coding rules
Most coding standards contain a hierarchy of rules according to their "strengths": some rules define things that are always required, things that are simply recommended, things that are allowed only with proper justification, things that are forbidden, etc.

However, it should be understood that there are always exceptions to the rules. In some cases, strict adherence to the rules even when they are not applicable can be dangerous, and lead to consequences worse than what the rule was intended to prevent (see section 4.3 "Undesirable effects").

Therefore, rather than defining "strong" or "weak" rules, it is better to state as a general principle that any rule can be violated in some special cases, provided that exceptions are properly justified, especially by showing that following the rule would lead to consequences that would oppose the intended purpose of the rule.

1.3 Some publicly available coding standards
When designing a new coding standard, it is advisable to start from a recognized source. Many companies have developed their own (and generally proprietary - if not company-secret) standard. However, some documents are freely available on the web, and can be used as a starting point:

- Ada Quality and Style Guide [1]: This document, initially designed for Ada83, then updated to Ada95, is not a coding standard per se, but rather a template for producing coding standards. It results from a careful study of best practices, and most coding standards use it
as a basis. It has been turned into a wiki [2] for the general public to help in evolving it to more recent versions of Ada.

- ESA Coding Standard [3]: this standard is a bit old (1998), and was mainly concerned with the transition from Ada83 to Ada95. A number of rules are questionable, and although it is an interesting reading, it should be taken with care as a starting point.

- NASA Coding Standard [4]: (actually, the Goddard Dynamic Simulator Coding Standard - not a general NASA standard). Developed by S. Leake, this document is permanently being updated, and addresses the latest versions of Ada. It features a number of interesting ideas.

2 Survey of some common coding rules

In this part, we provide some examples of commonly found rules, with a discussion why the rule might not be as obvious as it may seem at first sight - or even have adverse effects. All these examples are genuinely taken from actual coding standards - none was made up.

2.1 Examples of generally accepted rules

Every task must have an exception handler (that prints a message to Standard_Error, NASA 12.4). Tasks that terminate silently are the cause of very hard to find errors, and although the rule does not fully avoid silent termination of tasks, it forces the programmer to consider the issue - which is a good thing.

No return in procedures, only one in functions (Variant: only one return point). This is a common rule, since unnoticed returns from subprograms are a common cause of errors (especially when modifying the code). However, the rule as stated is not sufficient: it should consider also the case of exception handlers, which in fact create more return points.

2.3 Examples of debatable/debated issue

Many coding standards include lengthy rules about capitalization of identifiers or keywords, indentation, spacing etc. Although uniformity of presentation is important, it is questionable to enforce this in a coding standard, because checking the rule is about the same effort as fixing the presentation with a pretty-printer. It is actually simpler to just define a pretty-printer program (with appropriate parameterization) to be run on any source code.

There is no generally accepted naming convention. For example, some standards require type names to be terminated by "_T" or "_Type", while others forbid it. There is no clear accepted convention on such issues, except for one: whatever convention is required by the standard, it should be applied uniformly and consistently.

Usage of use clauses is also a highly debated issue. Some common rules are:

Use clauses are not allowed.

Use clauses are allowed only for predefined packages.

Use clauses shall be limited to the innermost scope where they are useful.

Use clauses are allowed in bodies, but not in specifications or context clauses.

This variety of rules reflect the fact that use clauses help readability by providing names that are easier to grasp and understand, at the cost of relying on visibility rules for making sure that the intended element is used. In the end, which rule to apply is a trade-off between safety, readability, and maintainability, which depends on the application domain.

2.4 Examples of inappropriate rules

Rules may be truly inappropriate, sometimes because they simply do not apply to Ada, or even sometimes because they originate from a clear misunderstanding of the language.

The use of the goto statement shall be avoided. This rule is found in almost every coding standard for Ada … and practically useless. The "goto debate" died long ago, and very few people are tempted to use a goto in Ada. This means that, if there is a goto in a program, chances are that it is justified. Therefore, prohibiting goto's in a coding standard is useless (although harmless in general, safe for the fact that it increases the number of rules).

Always use the short circuits form of boolean operators (and then, or else). Most of the time, this rule is guided by efficiency considerations. However, there is no evidence in general that short-circuits evaluations are more efficient than regular boolean operators. More generally, any rule motivated by efficiency considerations is at best applicable on a single version of a single compiler, and only after careful measurements to justify it. On the other hand, there is one good reason for this rule (but not related to efficiency): if full conditions coverage is required, short circuit operators reduce the testing effort, since they require only three combinations of values, while the regular operators require four.

Every "if" must have an "else" part. This is an example of an inappropriately imported rule from Misra/C, where it is expected to prevent an "else" to refer to the wrong "if". However, since Ada has an "end if", the problem it is expected to prevent in C simply cannot happen in Ada.

No 'Access since it causes dynamic allocation. This is a clear misunderstanding of the role of 'Access, since it actually prevents dynamic allocation!

Information shall be obtained from the compiler vendor as to the adopted method to support instantiation of generic units. Although this is good advice, it has nothing to do with the way code is written, and should not be part of a coding standard (but may well be in some other kind of document).

2.5 Rules with unexpected consequences

Those who design rules know well what they want achieve; however, the way the rules are formulated may lead to unexpected consequences, especially when enforced by a
tool that will apply the rule as stated, not necessarily as intended!

Predefined types shall not be used. This rule is intended to improve portability by not depending, typically, on the range of Integer and the like. But this formulation would forbid the types Character, String, and even Boolean!

All subprogram calls shall use named notation. This is a typical example of a rule that seems to make sense, but where it is quite difficult to capture the real intent. For example, a strict application of the rule would forbid:

\[ A*B + C*D \]

and require instead (since named notation is not possible with infix notation):

\[ \text{"+" (Left => "*" (Left => A, Right => B), Right => \text{"*" (Left => C, Right => D))} \]

So, should operators be exempted from the rule? What about calling subprograms with only one parameter, or whose type are all different, thus preventing risks in case of confusion of parameters?

No numeric literal other than 0 or 1, use named constants instead. At the very least, this rule should have an exception for the definition of named numbers. And the rule is actively harmful if applied to things like bit positions in representation clauses. And even with these exceptions, a strict application of the rule would prevent useful uses like writing \"X**2\", etc.

3 Defining a coding standard

We hope from the previous examples that the reader is convinced that defining a good, useful, coding standard is far from easy. Moreover, the appropriateness of rules depends on the application domain, and is hard to estimate beforehand. We propose therefore the following road-map for defining a coding standard:

- Start from a well-defined basis, and especially consider carefully the Ada Quality & Style [1] document.
- Gather prospective rules, from your own knowledge and from informal discussions with colleagues (and especially the ones who will be using the coding standard).
- Conduct a round of evaluation for each rule, especially considering the following issues:
  - What is the problem that this rule will prevent/minimize? (no rule without a well-defined purpose).
  - Is this rule really necessary? (avoid having too many rules).
  - What are the possible adverse or perverse effects of the rule? (avoid rules with drawbacks that overcome the benefits).
  - What are the cases where the rule should not be obeyed? (prepare derogations).
  - Is this rule automatically checkable? (uncheckable rules are just wishful thinking).
- Define the structure of the document:
  - Introduction explaining the purpose of the coding standard.
  - Statements of the rules.
  - Procedure to request a derogation, and to submit improvements/suggestions to the rules.

Each rule should have a convenient reference number, and be described with information such as:

- Statement of the rule.
- Motivation for the rule.
- Examples (do's and don't's).
- Exceptions (cases where not applicable).
- Tool used to check.

4 Using the coding standard

4.1 Checking the rules

Needless to say, a standard is of no value if it is not applied. Proper usage of a coding standard in a company needs addressing a number of issues:

- Make sure every person who is part of software development knows that the standard exists.
- Make sure that every concerned person has read and understood the standard.
- Make sure that the standard is perceived as a help rather than as an extra burden.
- Promote use of checking tools.

Note that the term "checking tools" encompasses various tools. The compiler itself may have options to check various aspects of the code (in addition to legality of course), and proper use of pragma Restriction can be handy for enforcing certain rules. Other than the compiler, there are various specialized tools, such as:

- Ada-Assured (Gramma Tech)
- AdaControl (Adalog)
- Adarc (RainCode)
- Gnatcheck (Adacore)
- Rule Checker (Logiscope)
- Testbed (LDRA)

Finally, peer reviews are the last "tool" for rules that cannot be automated, but they are hard to use systematically, and especially after each modification to the code, due to their high cost.

Another important issue is when to check the code. The general advice is "as early as possible". Some rules (like casing of identifiers) can be checked during typing with a properly parameterized syntactic editor. With a proper IDE, the checking tool should be integrated, therefore allowing the checking of units at the click of a button. This makes it easy for developers to check the rules after each successful compilation (many rules cannot be checked unless the code is compilable).
In addition (but preferably not in place!), checking the code should be part of the VCS process. Scripts, or hooks in the VCS, can be used to check units before they are entered in configuration. Refusing acceptance of units that do not follow the standard is a strong incentive for developers to obey by the rules.

An extra checking of the rule should be part of integration / acceptance tests. Checks should not fail at this point, since correcting flaws could be quite expensive. On the other hand, it is the only place where manual reviews can be practical.

And then... it's too late. Unfortunately, it happens sometimes that coding standards are defined very late in the development process, and applied to existing, sometimes already certified, code. Since any change would require running the certification process again, it is almost impossible to correct deficient code. Short of fixing it, it is then necessary to justify that every non-conformance is not a safety hazard. Quite a boring and costly process...

4.2 Obstacles to acceptance

Having the standard accepted by the developers is not an easy task. A number of psychological factors may come into play.

First of all, there might be a feeling that the standard has been designed by people "who don't know the issues". This happens often if the standard is imposed by the QA department, without proper interaction with the final users. Participation of the whole development team to the design of the coding standard is the first key to acceptance.

Another similar factor is the feeling that the rules are not appropriate to the kind of development ("that's not the rules we need!") or that they are actively harmful by preventing useful programming patterns ("those rules do not allow us to do it the way we want"). Note that the last case may or may not be justified; for example, a low level device driver may need extensive use of Unchecked_Conversion, but a careful examination may reveal that, safe for a small number of places, unchecked conversions can be replaced by higher level constructs.

Rules that are too difficult to check (especially those that require manual checking) may simply be too costly to check and therefore quickly become ignored.

Lack of proper information about the importance of early detection of violations may lead to a situation where the developers constantly delay the time to check ("we are under pressure to deliver, we'll check later"). And then when the checks are eventually performed, it is too late to fix the violations. Ease of automatic checking, and especially a convenient integration of the checking tool into the IDE is necessary to avoid this effect.

4.3 Undesirable effects

It must be stressed that obeying the coding standard cannot be an excuse for writing bad code! Typically, we discovered in a project review a lot of places that used variables overlays to overcome the typing system - in contexts where an Unchecked_Conversion (which would have been much safer) would have been perfectly justified. When asked why these overlays were used, the developers said "well, Unchecked_Conversion was not allowed, so we had to find something else".

The real issue here is that the developers felt easier to exploit an omission in the rules (overlays should have been forbidden as well) than to ask for a derogation, which would have been perfectly justified.

4.4 Coding standard and legacy code

It is a fact of life that coding standards are often designed at a time where a lot of code exists, and is being reused even in new projects. There is little chance that legacy code meets spontaneously the standard, and fixing it would require a lot of effort, and even induce a risk for certified software.

In this case, the best solution is to take the opportunity of a change for some other reason to bring the software back to conformance, i.e. don't touch it until it has to be modified, but when modified, take the extra move to fully fix it.

Conclusion

Introducing a coding standard can be deemed successful if the standard is well accepted, actually applied project-wide, and results in an improvement of the quality of the product.

These goals require a reasonably sized set of rules that can be easily checked with automated tools. The rules must be designed in cooperation with the future users who must be aware that the design of appropriate rules is an iterative process where their feedback is welcome.

Information of users must include justifications of when the rules are to be applied or not, and that granting derogations when appropriate is part of the normal process.

References

   http://www.adaic.org/resources/add_content/docs/95style/95style.pdf


Advanced Ada Support for Real-Time Programming *

Mario Aldea Rivas
Universidad de Cantabria, 39005 Santander, Spain; email: aldeam@unican.es

Abstract

This paper is an extended summary of the tutorial given at Ada-Europe 2012.

In the 2005 and 2012 revisions of the Ada standard, real-time programming has experienced a large improvement but most of the new services introduced are unknown or underused due to the lack of free software implementations. The tutorial presented an overview of these new services trying to focus on their utility for real-time systems and their typical use patterns.

Keywords: Ada 2005, Ada 2012, real-time systems, programming languages.

1 Introduction

The support of Ada for real-time programming has experienced a large improvement in the last years positioning the language one step ahead of other real-time languages and operating system interfaces. Functionalities such as hierarchical scheduling based on priority ranges, new dispatching policies, execution time clocks and timers, timing events, etc. are in the standard from the 2005 revision [1] [2] but they have not gotten the relevance that they deserve due to the lack of free software implementations.

Most of these relatively new services are starting to be available in some platforms. In particular, in this tutorial we used the MaRTE OS/GNAT [3] platform. This platform supports most of these new services [4] [5]:

- Timing events.
- Execution time clocks and timers.
- Task group execution time budgets.
- Dynamic ceiling priorities for protected objects.
- Additional scheduling policies: Round robin, EDF, Mixed (priority-specific policies).
- Immediate priority changes.
- Execution Time for Interrupt Handlers.

2 Classic Ada real-time model

In the Ada 95 real-time model the task is the concurrency unit and the synchronization and mutual exclusion among tasks is accomplished by the use of protected objects or rendezvous. Analysable scheduling is achieved with the use of the FIFO_Within_Priorities dispatching policy and the locking policy Ceiling_Locking.

Time management, and in particular the periodic activation of tasks, is performed with the monotonic clock provided by package Ada.Real_Time and the delay until statement. Other services related to real-time in Ada 95 are the dynamic priorities for tasks and the interrupt handling facilities among others.

The classic model allowed to dynamically change the priority of the tasks, but there was not a similar functionality to...
Advanced Ada Support for Real-Time Programming

<table>
<thead>
<tr>
<th>Ada 95</th>
<th>Ada 2005 additions</th>
<th>Ada 2012 additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>Ravenscar profile</td>
<td>Multiprocessor and Dispatching domains</td>
</tr>
<tr>
<td>FIFO_Within_Priorities</td>
<td>Timing events</td>
<td>Group budgets and multiprocessors</td>
</tr>
<tr>
<td>Dynamic priorities</td>
<td>Execution time clocks and timers</td>
<td>Barriers</td>
</tr>
<tr>
<td>Protected objects</td>
<td>Group budgets</td>
<td>Suspension objects</td>
</tr>
<tr>
<td>Ceiling_Locking</td>
<td>Non-preemptive dispatching</td>
<td>“Synchronization” aspect</td>
</tr>
<tr>
<td>Monotonic clock</td>
<td>EDF dispatching</td>
<td>Yield_To_Higher</td>
</tr>
<tr>
<td>delay until</td>
<td>Round robin dispatching</td>
<td>Execution time of interrupt handlers</td>
</tr>
<tr>
<td></td>
<td>Priority specific dispatching</td>
<td>Suspend_Until_True_And_Set_Deadline</td>
</tr>
<tr>
<td></td>
<td>Dynamic priorities for POs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Synchronized interfaces</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Partition elaboration policy</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Evolution of the real-time Ada

Listing 1: Use of the Priority attribute

```ada
protected body PO is
    procedure Change_Ceiling (Prio: in System.Priority) is
    begin
        ...  -- PO'Priority has old value here
        PO'Priority := Prio;
        ...  -- PO'Priority has new value here
    end Change_Ceiling;  -- ceiling is changed here
    ...
end PO;
```

change the ceiling of the protected objects. This was a serious limitation in many applications, specially in those that require “mode changes”. This problem was solved in Ada 2005 with the definition of the attribute Priority for the protected objects (see Ada 2012 Reference Manual2 D.5.2). An example of use of this attribute is shown in Listing 1. Note the unusual syntax: it is the only attribute in the language that is possible to assign a value to.

Other addition to the classic model is the introduction in Ada 2005 of the Detect_Blocking pragma (RM H.5). Its use in a partition forces to detect potentially blocking operations within any protected action.

The introduction of the “aspects” in Ada 2012 has had an important impact everywhere in the language. In relation to the real-time concurrency model the most important effect is the change in the assignment of the priority to tasks and protected objects. Now aspects should be used for this purpose instead of the old pragmas (that are declared obsolescent):

```ada
task Controller with Priority => 12;

protected PO with Priority => 20 is
    ...
end PO;
```

Other very relevant addition in Ada 2005 was the Ravenscar Profile (RM D.13). A profile is a collection of restrictions and other pragmas that describes a subset of the language intended for a particular purpose. The Ravenscar Profile is a subset of the Ada tasking model targeted to critical real-time applications. The main objectives of this profile are:

- Produce a deterministic concurrent execution model that can be analysable.
- Allow an efficient and small implementation of the runtime library.

Most of the restrictions of the profile have the objective of produce static applications where all the tasks and protected objects are defined at library level and tasks never terminate. The profile also avoids constructs that can be very complex or difficult to analyse like the abort, select or requeue statements.

3 Advanced time management

In Ada 95 the real-time management was based on the monotonic clock defined in the Ada.Real_Time package and on the delay and delay until statements. Nowadays Ada provides a much larger diversity of services for time management with the definition of new clocks and timers:

- Execution time clocks for tasks (Ada 2005).
- Execution time clocks for interrupt handlers (Ada 2012).
- Execution time timers for tasks (Ada 2005).
- Execution time timers for groups of tasks (Ada 2005).

3.1 Timing events

Package Ada.Real_Time.Timing_Events (RM D.15) allows user-defined handlers to be executed at a specific time. The handler is a protected procedure that is executed at interrupt priority directly by the system timer interrupt service routine without the need of using an auxiliary task or a delay statement.

They are intended for applications that require to execute a short action at a very precise time. A typical example would be a control system where the output to the actuators must be updated at a very precise rate. They are also useful to implement scheduling algorithms that requires programming scheduling actions to be done at a future point in time.

As a simple example, Listing 2 shows the body of a protected object used to generate a periodic pulse based on the use of a timing event.
The main application of these timers is to take corrective actions on WCET overrun situations. This situations are relatively common in modern architectures since pipelines, branch prediction, cache effects, and so on, makes it very complex to measure the actual WCET of a task. Using the execution time timers, the corrective action (lower the task priority, enter in a safe operation mode, etc.) can be done by the timer handler at the very moment the task overruns its WCET.

### 3.4 Group execution time budgets

The package Ada.Execution_Time.Group_Budgets (RM D.14.1) allows to assign execution time budgets to a group of tasks (with the restriction that a task can only belongs to one group). The execution of any task member of the group results in the budget counting down. When the budget becomes exhausted, the user-defined handler (a protected procedure) is executed.

The main application of the group budgets is the implementation of “aperiodic servers” [8] to achieve temporal isolation among different parts of a complex application where each part, maybe an independent application, is made up of a number of tasks.

With the multiprocessor support provided in Ada 2012, the group budget definition has been tuned and now a group budget is attached to a particular processor. Only execution of the tasks in this particular processor reduces the remaining budget of the group.

### 3.5 Execution time of interrupt handlers

A common assumption is that the effect of the interrupt handlers on the execution time clocks of the tasks is negligible because handlers are usually very short pieces of code. Under that assumption systems charge the time consumed by the interrupt handlers to the task executing when the interrupt is generated.

This assumption may not be realistic in real-time systems that undertake an intensive use of interrupts or uses timing events with relatively long handlers. In these systems, it would be desirable to have a separate account of the interrupt handlers execution time.

The Ada language, in the packages Ada.Execution_Time and Ada.Execution_Time.Interrupts (RM D.14.3) provides support for the separate accounting of the execution time of interrupt handlers. This time includes the time consumed by the timing event handlers since they are executed directly by the system timer ISR.

The RM allows to the implementations to provide three support levels:

1. No support at all: “it is implementation defined which task, if any, is charged the execution time that is consumed by interrupt handlers” (RM D.13,11/3).
2. Execution time of interrupt handlers not charged to tasks and accounted by one global clock.
3. Execution time of interrupt handlers not charged to tasks and accounted by one different clock for each interrupt.

The support level is defined by the value of the boolean constants Interrupt_Clocks_Supported and Separate_Interrupt_Clocks_Supported defined in the package Ada.Execution_Time.

### Listing 2: Periodic pulse generator based on a timing event

```ada
-- Timing event declaration
Pulse : Timing_Event;

protected body Pulser is
    procedure Start is
        begin
            Output_High;
            Next_Time := Clock + Pulse_Interval;
        end Start;

    procedure Stop is
        Cancelled : Boolean;
        begin
            Cancel_Handler(Pulse, Cancelled);
            if not Cancelled then
                raise Handler_Not_Set;
            end if;
        end Stop;

    -- This is the handler of the timing event
    procedure Handler (T : in out Timing_Event) is
        begin
            Output_Swap;
            Next_Time := Next_Time + Pulse_Interval;
        end Handler;

end Pulser;
```

### 3.2 Execution time clocks

Each task has an associated execution time clock (package Ada.Execution_Time, RM D.14) which measures its execution time, that is, the time spent by the system executing that task. Having such clocks eases the measurement of the worst-case execution times (WCET) of the tasks, a key parameter in the schedulability analysis of the real-time applications.

Besides their usefulness for measuring the WCET, the execution time clocks are also very important in real-time systems because they are the base clocks of the execution time timers, as it will be described in Section 3.3.

### 3.3 Execution time timers

The execution time timers (package Ada.Execution_Time.Timers, RM D.14.1) allow user-defined handlers to be executed when the execution time clock of a task has reached the desired value. From the user’s interface point of view, they look very much like the timing events, both have an expiration time and a protected handler procedure.

The main application of these timers is to take corrective actions on WCET overrun situations. This situations are relatively common in modern architectures since pipelines, branch prediction, cache effects, and so on, makes it very complex to measure the actual WCET of a task. Using the
4 Advanced dispatching

In Ada 95 there was only one predefined dispatching policy called FIFO_Within_Priorities. It defines a fixed priority scheduling with FIFO order for tasks with the same priority.

In Ada 2005 three new dispatching policies where defined:

- Non_Preemptive_FIFO_Within_Priorities (RM D.2.4): fixed priority without preemption when a higher priority task is runnable.
- Round_Robin_Within_Priorities (RM D.2.5): fixed priority with cyclic scheduling between tasks with the same priority.
- EDF_Across_Priorities (RM D.2.6): “Earliest Deadline First” scheduling policy.

The policies can be applied to the whole partition using the configuration pragma:

```
pragma Task_Dispatching_Policy (dispatching_policy);
```

They can also be applied to a particular priority range using pragma Priority_Specific_Dispatching. The priority specific dispatching will be described in Section 4.4.

4.1 Non-Preemptive dispatching

The policy identifier Non_Preemptive_FIFO_Within_Priorities defines a policy identical to FIFO_Within_Priorities but without preemption when a higher priority task is runnable. When this policy is in use, a task will run until completion or until it is blocked or executes a delay statement (More technically: the only dispatching points are blocking or termination of a task, a delay, or a call to a “yield” procedure).

This policy is intended to be used in high-integrity applications since it is much more deterministic than preemptive policies and it is a intermediate step between cyclic executives and preemptive multitasking.

Since non-preemption reduces schedulability, it is usual when using a non-preemptive dispatching that tasks volunteer to be preempted at some points of its execution. Traditionally Ada tasks yields the CPU using a delay 0.0 statement.

In Ada 2012 new yield procedures has been added to packages Ada.Dispatching and Ada.Dispatching.Non_Preemptive (RM D.2.4). The most interesting of this new yield procedures is Yield_To_Higher, this procedure only yields the CPU to tasks with higher priority than the calling task. An interesting point about this procedure is that it can be used from inside a protected action, since if the Ceiling_Locking policy is in use the possible preemption cannot put in danger the mutual exclusion achieved by the protected object.

4.2 Round Robin dispatching

The policy Round_Robin_Within_Priorities allows a set of tasks with the same priority to make progress at a similar rate:

1. Each task can execute at most during an interval of time called “quantum”.
2. When the quantum is exhausted, and the task is not executing a protected operation, it is moved to the tail of its priority queue.
3. The task at the head of the priority queue gets the CPU.

This policy is usually applied to mixed dispatching applications, where the tasks with real-time constrains are dispatched under FIFO or EDF policies at the highest priority levels, and the non real-time tasks are executed at the lowest priority under the Round Robin policy, in order to share the spare time.

The package Ada.Dispatching.Round_Robin allows to get and set the quantum assigned to each priority level where the Round robin policy is applied. Note that the RM allows to the implementations to restrict the available quantum values and, in consequence, the quantum assigned by the programmer (using procedure Set_Quantum) could be different from the one that is actually been used by the implementation (returned by Actual_Quantum).

4.3 Earliest Deadline First dispatching

The EDF policy (EDF_Across_Priorities) is the most popular dynamic priority policy. It is based on the concept of “deadline”, that is, the time when an activation of a task should have finished its job.

The policy requires a new scheduling attribute to be defined for the tasks: the “relative deadline”. For each activation a task has a different “absolute deadline” that is equal to its activation time plus its relative deadline. Tasks at the same priority level are ordered according to their absolute deadlines (the task with the earliest absolute deadline is executed first).

Scheduling theory proves that dynamic priority policies allow a better resource usage in some cases. EDF is the most popular dynamic priority policy for several reasons:

- It's implementation is relatively simple compared to other dynamic priority policies.
- It is optimal in monoprocessors: if a set of tasks is schedulable by any dispatching policy then it will also be schedulable by EDF.
- It can guarantee all the tasks' deadlines at higher processor load (up to 100%) than fixed priorities.

Of course, EDF has disadvantages compared to fixed priority policies:

- It requires a more complex implementation than the fixed priority policies what implies a higher scheduler overhead.
- Under an overload situation the set of tasks that will miss their deadlines is unpredictable.
4.3.1 Managing EDF tasks

The initial relative deadline of a task can be specified with the Relative_Deadline aspect (pragma Relative_Deadline is declared obsolete):

```ada
task EDF_Task with Relative_Deadline => Time;
```

The Time value is an expression of type Real_Time.Time_Span. The first absolute deadline of the task will be its activation time plus the relative deadline assigned with this aspect. If the aspect is not specified, then the initial absolute deadline of a task is Ada.Real_Time.Time_Last.

The package Ada.Dispatching.EDF provides operations to set and get the absolute deadline of a task. It also provides the procedure Delay_Until_And_Set_Deadline mainly intended to create periodic EDF tasks:

```ada
-- Periodic EDF task with deadline equal to period
task body Periodic_Task is
   Interval : Time_Span := Milliseconds (10);
   begin
      Next : Time;
      loop
         Next := Clock; -- Start time
         Next := Next + Interval;
      end loop;
   end Periodic_Task;
```

The call to Delay_Until_And_Set_Deadline delays the task until the time Next and, when the task becomes runnable again, it will have an absolute deadline equal to Next plus Interval. The use of this procedure avoids unnecessary context switches that can happen if the Set_Deadline procedure and the delay until statement are used instead.

4.3.2 EDF and the priority ceiling protocol

The definition of the Ceiling_Locking policy suffers some changes when applied to EDF tasks. In such situation, the protocol defined by the Ada RM is know in the literature as the “Immediate Ceiling Priority Protocol” (ICPP) (also known as “Baker’s Protocol” or “SRP Protocol”) [9]. This protocol is a generalization of the “Immediate Ceiling Priority Protocol” (ICPP) the Ada interpretation of the Ceiling_Locking policy for FIFO tasks.

The PLCP requires a new parameter for tasks and protected objects: the “preemption level”. In the definition of the PLCP, the preemption level is a small integer number that should be assigned to the tasks in deadline monotonic order, the shorter the relative deadline of a task, the higher its preemption level. The preemption level of a protected object is the maximum preemption level of any task that uses it.

In the Ada definition of PLCP the priority of tasks and protected objects is used in the role of the preemption level. So, the declaration of two EDF tasks could be:

```ada
task EDF_Task_With_Short_Deadline with
   Relative_Deadline => Ada.Real_Time.Milliseconds (10),
   Priority => 4;
```

```ada
task EDF_Task_With_Long_Deadline with
   Relative_Deadline => Ada.Real_Time.Milliseconds (20),
   Priority => 3;
```

The rules to integrate the PLCP in the Ada priority based model are quite complex (see RM D.2.6, 23/2-26/3) but they are only relevant for implementers. The programmer only needs to care about setting the preemption level (priority) of tasks and protected objects as explained above.

4.4 Mixed hierarchical dispatching

Ada goes a step further in dispatching flexibility by supporting mixed hierarchical scheduling configurations. A two-levels dispatching model is defined with a base fixed priority policy and several second-level dispatching policies in non-overlapping priority ranges.

The configuration pragma

```ada
Priority_Specific_Dispatching is used for this purpose:
```

```ada
pragma Priority_Specific_Dispatching (policy_identifier, first_priority, last_priority);
```

By using this pragma, tasks with active priority in the range [first_priority, last_priority] are scheduled under the policy specified by policy_identifier (where policy_identifier can be any Ada dispatching policy but the Non-Preemptive which can only be used as the global partition dispatching policy).

When several priority ranges are defined, high priority ranges take precedence over low priority ranges according to the key rule of the base fixed priority policy: the processor is assigned to the first task of the highest occupied priority queue.

A task can “jump” from one priority range to another when its base priority is changed (using package Dynamic_Priorities) or while it is inheriting a priority. Special care has to be taken when, due to a base priority change, a task “jumps” to an EDF range. In such situation, and in the case the Relative_Deadline aspect was not specified for the task, it will have the longest absolute deadline and, consequently, it will be the less prioritary task in the range.

Protected objects can be used to share data between tasks in different priority ranges. As it could be expected, the
Ada multiprocessor support is intended for "Symmetric multiprocessing" (SMP). In a SMP architecture two or more processors are connected to a single shared memory.

Mixed scheduling allows to combine in the same application the good properties of the different policies. For example, with the configuration described in Listing 3 an application could take advantage of the predictability of the FIFO scheduling for the critic tasks, the better resource usage provided by EDF for the non-critic tasks and the fair distribution of resources provided by the Round robin policy for the non-RT tasks.

Listing 3: Priority ranges configuration

```ada
pragma Priority_Specific_Dispatching (FIFO_Within_Priorities, 10, 16);
pragma Priority_Specific_Dispatching (EDF_Across_Priorities, 2, 9);
pragma Priority_Specific_Dispatching (Round_Robin_Within_Priorities, 1, 1);
```

Ceiling_Locking rules are obeyed and the promoted task competes with the other tasks in the range according to the priority it has just inherited.

5 Multiprocessor support

Multiprocessor architectures are becoming popular in many application areas including the embedded systems. Ada is ready to face this important architectural change thanks to the new services defined in Ada 2012. The core of the new Ada multiprocessor support are the “Dispatching Domains”, other services like the “Synchronous Barriers” are also targeted to multiprocessor architectures.

Ada multiprocessor support is intended for “Symmetric multiprocessing” (SMP). In a SMP architecture two or more identical processors are connected to a single shared memory.

Package System.Multiprocessors (RM D.16) defines the integer type to identify the processors and also provides a function to know the number of processors in the system.

5.1 Dispatching domains

The package System.Multiprocessors. Dispatching Domains (RM D.16.1) allows to group processors into “Dispatching domains”. Each domain is a contiguous range containing one or more processors. Each processor belongs to only one dispatching domain.

At the beginning of the execution all the processors belong to the System_Dispatching_Domain and the environment task is allocated to it.

During the elaboration of the partition the programmer can create new domains that will remain unchanged during the rest of the execution of the application. As processors are added to the new dispatching domains they are removed from the System_Dispatching_Domain. Dispatching domains are created using the Dispatching_Domains.Create function:

```ada
Domain_1 : Dispatching_Domain := Create (15, 17);
```

Every task is allocated to a dispatching domain. Inside its domain, a task can execute in any processor unless it is explicitly assigned to a particular processor. The processor affinity of a task inside its domain can be changed at run-time as many times as desired.

This flexibility allows Ada to support the most popular allocation approaches:

1. Fully Partitioned: each task is allocated to a single processor on which all its jobs must run.
2. Dynamically Partitioned: at run-time the application can change the assignment of a task from one processor to another.
3. Partially Partitioned: tasks are restricted to a subset of the available CPUs, jobs may migrate during execution.
4. Global: all tasks/jobs can run on all processors, jobs may migrate during execution.

By default all the tasks are allocated to the System_Dispatching_Domain. A task can be allocated to a user defined dispatching domain using the Dispatching_Domain and CPU aspects:

```ada
Allocate task to Domain_1 (the task can execute in any processor in the domain)
```

```ada
Allocate task to Domain_1 and assign it to the processor 16
```

Alternatively, a task allocated to the System_Dispatching_Domain can be allocated to a user defined domain with the Dispatching_Domains.Assign_Task procedure:

```ada
Allocate task to Domain_1 (the task can execute in any processor in the domain)
```

```ada
Assign Task (Domain_1, T'Identity);
```

Note that Assign_Task can only be used to move tasks from the System_Dispatching_Domain. Once a task has been allocated to a user defined domain it will remain in that domain forever.

At any time we can use the the Set_CPU procedure to change the affinity of a task inside its domain:

```ada
Assign task to the processor 17
```

```ada
Set_CPU (17, T'Identity);
```

During the revision process it was considered to include in the Ada standard the possibility of specifying dispatching policies on a per-dispatching domain basis.

Although this functionality was finally rejected for being considered too complex, there is a less elegant but efficient approach that can be used. This approach consist on including in a dispatching domain tasks in a specific priority range and use the Priority_Specific_Dispatching pragma to apply the desired dispatching policy to that range and consequently to the tasks in the dispatching domain.
5.2 Synchronous Barriers

The synchronous barriers (package Ada.Synchronous_Barriers, RM D.10.1) are a synchronization primitive intended for massively parallel machines. To take advantage of the parallelism provided for such architectures, it is usual to use algorithms that can be performed in parallel for a large number of tasks. After this parallel part, it is very common that the algorithm has a final sequential part to recombine the results.

Usually the parallel computations are very short. In that case the use of a complex synchronization primitive would remove any gains obtained from the use of the parallel algorithm. Synchronous barriers have been designed to solve this problem since they can be implemented very efficiently.

Synchronous barriers are used to synchronously release a group of tasks after the number of blocked tasks reaches a specified count value.

A barrier is created specifying is “release threshold” (the number of blocked tasks required for the barrier to be open). When a task reaches the barrier (calls the Wait_For_Release procedure) it is blocked in the barrier.

procedure Wait_For_Release (The_Barrier : in out Synchronous_Barrier; Notified : out Boolean);

If the number of blocked tasks reaches the release threshold the barrier is open an all the tasks are released. Only one of the released tasks will be notified with the Notified parameter set to True. In the case that a final sequential part of the algorithm is required, the programmer can use this notifications to be sure that one, and only one, task will do this final part of the computation.

References


Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/adanswers/gems.

Gem #132: Erroneous execution - Part 1
Bob Duff, AdaCore

Abstract. Many Ada programmers are confused by the term "erroneous", in part because Ada uses the term to mean something other than what it means in plain English. In English, "erroneous" just means "wrong". But in Ada it refers to a particular kind of wrongness. This gem is intended to clarify the Ada meaning.

Let's get started...

Ada is pretty good about requiring compilers to detect errors at compile time, or failing that, at run time. However, there are some kinds of errors that are infeasible to detect. The Reference Manual calls such errors "erroneous execution".

RM-1.1.5(10) defines the term:

"...[T]he implementation need not detect such errors either prior to or during run time. ...[T]here is no language-specified bound on the possible effect of erroneous execution; the effect is in general not predictable."

An example of erroneous execution is suppressing a check that fails. For example:

```ada
pragma Suppress (All_Checks); -- Or use the -gnatp switch

A (X) := A (X) + 1;
```

If X is out of bounds, then the above results in erroneous execution. That means that the program can do anything at all. In explaining the meaning of erroneousness, people often like to talk of spectacular disasters: "It might erase your system disk!" "Your keyboard might catch on fire!" "Nasal demons!"

I think that's somewhat misleading. For one thing, if you're running under an operating system, with proper protections set up, erroneous execution will not erase your system disk. The point is that Ada doesn't ensure that, but the operating system does. Likewise, Ada doesn't prevent your keyboard from catching on fire, but we hope the computer manufacturer will.

One disaster that actually might happen is that the above code will overwrite some arbitrary memory location. Whatever variable was stored there might be destroyed. That's a disaster because it can take hours or even days to track down such bugs. If you're lucky, you'll get a segmentation fault right away, making the bug much easier to figure out.

But the worst thing of all is not keyboard fires, nor destroyed variables, nor anything else spectacular. The worst thing an erroneous execution can cause is for the program to behave exactly the way you wanted it to, perhaps because the destroyed memory location wasn't being used for anything important. So what's the problem? If the program works, why should we care if some pedantic language lawyer says it's being erroneous?

To answer that question, note this common debugging technique: You have a large program. You make a small change (to fix a bug, to add a new feature, or just to make the code cleaner). You run your regression tests, and something fails. You deduce that the cause of the new bug is the small change you made. Because the change is small relative to the size of the whole program, it's easy to figure out what the problem is.

With erroneousness, that debugging technique doesn't work. Somebody wrote the above erroneous program (erroneous if X is out of bounds, that is). It worked just fine. Then a year later, you make some change totally unrelated to the "A (X) := A (X) + 1;" statement. This causes things to move around in memory, such that now important data is destroyed. You can no longer assume that your change caused the bug; you have to consider the entire program text.

The moral of the story is: Do not write erroneous programs.

Gem #133: Erroneous execution - Part 2
Bob Duff, AdaCore

Abstract. The previous Gem said that "erroneous execution" means that anything can happen as far as the Ada RM is concerned, and in particular, the program might work properly. This Gem continues the discussion.

Let's get started...

The moral of the story was: Do not write erroneous programs.

Strictly speaking, that's wrong usage. "Erroneous" refers to a particular execution of a program, not to the program itself. It's possible to write a program that has erroneous behavior for some input data, but not for some other input data. Nonetheless, it's reasonable to use "erroneous program" to refer to a program that might have erroneous behavior. Just remember that "erroneous" is not a property of the program text, but a property of the program text plus its input, and even its timing.

Never deliberately write code that can cause erroneous execution. For example, I've seen people suppress Overflow_Checks, because they "know" the hardware does wrap-around (modular) arithmetic, and that's what they want. That's wrong reasoning. The RM doesn't say that overflow, when suppressed, will do what the hardware does. The RM says anything at all can happen.

If you suppress Overflow_Checks, you are telling the compiler to assume that overflow will not happen. If overflow can happen, you are telling the compiler to assume a falsehood. In any mathematical system, if you assume "false", anything at all can be proven true, causing the whole house of cards to tumble. Optimizers can and do prove all sorts of amazing things when told to assume "false".
Never try to guess that the optimizer isn't smart enough to cause trouble. Optimizers are so complicated that even their authors can't accurately predict what they will do. For example:

\[
X : \text{Natural} := \ldots;
Y : \text{Integer} := X + 1;
\]
\[
\text{if } Y > 0 \text{ then }
\text{Put\_Line (Y'Img);}\text{ end if;}
\]

The above will print some positive number, unless X is \text{Integer}'Last, in which case \text{Constraint\_Error} will be raised. The optimizer is therefore allowed to deduce that when we get to the 'if', Y must be positive, so it can remove the 'if', transforming it to:

\[
X : \text{Natural} := \ldots;
Y : \text{Integer} := X + 1;
\text{Put\_Line (Y'Img);}\]

That's good: removing the 'if' probably makes it run faster, which is the optimizer's goal. But if checks are suppressed, the optimizer can still do the above transformation. The reasoning is now: "when we get to the 'if', either Y must be positive, or we must be erroneous". If the former, the 'if' can be removed because it's True. If the latter, anything can happen (it's erroneous!), so the 'if' can be removed in that case, too. "X + 1" might produce \(-2^{31}\) (or it might not).

We end up with a program that says:

\[
\text{if } Y > 0 \text{ then }
\text{Put\_Line (Y'Img);}\text{ end if;}
\]

and prints a negative number, which is a surprise.

Another possible behavior of the above code (with checks suppressed) is to raise \text{Constraint\_Error}. "Hey, I asked for the checks to be suppressed. Why didn't the compiler suppress them?" Well, if the execution is erroneous, anything can happen, and raising an exception is one possible "anything". The purpose of suppressing checks is to make the program faster. Do not use pragma \text{Suppress} to suppress checks (in the sense of relying on not getting the exception). Compilers do not remove suppressed checks if they are free -- for example, imagine a machine that automatically traps on overflow.

Perhaps pragma \text{Suppress} should have been called something like \text{Assume\_Checks\_Will\_Not\_Fail}, since it doesn't (necessarily) suppress the checks.

**Gem #134: Erroneous execution - Part 3**

**Bob Duff, AdaCore**

**Abstract:** This Gem expands on the example of erroneous execution discussed in Part 2.

**Let's get started...**

We showed how this:

\[
X : \text{Natural} := \ldots;
Y : \text{Integer} := X + 1;
\]
\[
\text{if } Y > 0 \text{ then }
\text{Put\_Line (Y'Img);}\text{ end if;}
\]

can end up printing a negative number if checks are suppressed.

In fact, a good compiler will warn that "Y > 0" is necessarily True, so the code is silly, and you can fix it. But you can't count on that. Optimizers are capable of much more subtle reasoning, which might not produce a warning. For example, suppose we have a procedure:

\[
\text{procedure Print\_If\_Positive (Y : Integer) is}
\begin{align*}
& \text{begin} \\
& \text{if } Y > 0 \text{ then} \\
& \text{Put\_Line (Y'Img);} \\
& \text{end if;}
& \text{end Print\_If\_Positive;}
\end{align*}
\]

It seems "obvious" that Print\_If\_Positive will never print a negative number. But in the presence of erroneousness, that reasoning doesn't work:

\[
X : \text{Natural} := \ldots;
Y : \text{Integer} := X + 1;
\]
\[
\text{Print\_If\_Positive (Y);}\]

The optimizer might decide to inline the call, and then optimize as in the previous example.

Other language features that can cause erroneous execution include:
- Shared variables (erroneous if multiple tasks fail to synchronize)
- Address clauses
- unchecked\_Conversion
- Interface to other languages
- Machine-code insertions
- User-defined storage pools
- unchecked\_Deallocation

A complete list can be found by looking up "erroneous" in the RM index, or by searching the RM. Every case of erroneous execution is documented under the heading "Erroneous Execution".

You should try to minimize the use of such features. When you need to use them, try to encapsulate them so you can reason about them locally. And be careful.

As for suppressing checks: Don't suppress unless you need the added efficiency and you have confidence that the checks won't fail. If you do suppress checks, run regression tests on a regular basis in both modes (suppressed and not suppressed).

The final part of this Gem series will explain the rationale behind the concept of erroneous execution in Ada.

**Gem #135: Erroneous execution - Part 4**

**Bob Duff, AdaCore**

**Abstract:** This Gem completes the series on erroneous execution by discussing the language design. Why does Ada have erroneous execution in the first place?
Let's get started...

Many programmers believe that "optimizers should not change the behavior of the program". Many also believe that "optimizers do not change the behavior of the program". Both beliefs are false in the presence of erroneousness.

So if erroneousness is so bad, why does the Ada language design have it? Certainly, a language designer should try to minimize the amount of erroneousness. Java is an example of a language that eschews erroneousness, but that comes at a cost. It means that lots of useful things are impossible or infeasible in Java: device drivers, for example. There is also an efficiency cost. C is an example of a language that has way too much erroneousness. Every single array-indexing operation is potentially erroneous in C. (C calls it "undefined behavior").

Ada is somewhere in between Java and C in this regard. You can write device drivers in Ada, and user-defined storage pools, and other things that require low-level access to the machine.

But for the most part, things that can cause erroneousness can be isolated in packages -- you don't have to scatter them all over the program as in C.

For example, to prevent dangling pointers, try to keep the "new" and Unchecked_Deallocations together, so they can be reasoned about locally. A generic Doubly_Linked_List package might have dangling pointer bugs within itself, but it can be designed so that clients cannot cause dangling pointers.

Another way to prevent dangling pointers is to use user-defined storage pools that allow deallocation of the entire pool at once. Store heap objects with similar lifetimes in the same pool. It might seem that deallocating a whole bunch of objects is more likely to cause dangling pointers, but in fact just the opposite is true. For one thing, deallocating the whole pool is much simpler than walking complicated data structures deallocating individual records one by one. For another thing, deallocating en masse is likely to cause catastrophic failures that can be fixed sooner rather than later. Finally, a user-defined storage pool can be written to detect dangling pointers, for example by using operating system services to mark deallocated regions as inaccessible.

Note that Ada 2012 has "Subpools", which make user-defined storage pools more flexible.

A final point about erroneousness that might be surprising is that it can go backwards in time. For example:

```ada
if Count = 0 then
   Put_Line ("Zero");
end if;
Something := 1 / Count; -- could divide by zero
```

If checks are suppressed, the entire 'if' statement, including the Put_Line, can be removed by the optimizer. The reasoning is: If Count is nonzero, we don't want to print "Zero". If Count is zero, then it's erroneous, so anything can happen, including not printing "Zero".

Even if the Put_Line is not removed by the compiler, it can appear to be, because the "Zero" might be stored in a buffer that never gets flushed because some later erroneousness caused the program to crash.

Every statement about Ada must be understood to have ", unless execution is erroneous" after it. In this case, "Count = 0 returns True if Count is zero" is obviously true, but it really means "Count = 0 returns True if Count is zero, unless execution is erroneous, in which case anything can happen".

Moral: Take care to avoid writing erroneous programs.
National Ada Organizations

Ada-Belgium

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

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

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

Ada-France

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

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

Ada in Sweden

Ada-Sweden
attn. Rei Strähle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch