ADA USER JOURNAL

Volume 26
Number 4
December 2005

Contents

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 first 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 in Europe and other parts of the world. The language of the journal is English.

Although the title of the Journal refers to the Ada language, any related topics are welcome. In particular papers in any of the areas related to reliable software technologies.

The Journal publishes the following types of material:

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

Further details on our approach to these are given below.

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

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. Since not all of our readers have access to resources such as the World Wide Web and Usenet, or have enough time to search through the information that can be found in those resources, we reprint or report on items that may be of interest to them.

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.
We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal.*

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

All material for publication should be sent to the Editor, preferably in electronic format. The Editor will only accept typed manuscripts by prior arrangement.
Prospective authors are encouraged to contact the Editor by email to determine the best format for submission. Contact details can be found near the front of each edition.
Example papers conforming to formatting requirements as well as some word processor templates are available from the editor. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

The attentive reader will remember that, exactly one year ago, issue 25-4 of the journal featured the first instalment of the Rationale for Ada 2005, written by John Barnes and published as a primer by the Ada User Journal out a project funded by Ada-Europe. The journal issue I am very proud to close with this editorial contains the last two instalments of the project, of course along with other worthwhile material.

In December 2004 I was an easy prophet when I anticipated that the year 2005 would be gratifyingly rich with Ada-related events. In fact it surely was. And the Ada User Journal has happily ridden on this wealth by bringing to you 66 more pages worth of material in News, Calendars, Forthcoming Events and technical articles – including a full Rationale! I hope you did enjoy reading as much as I did working on the issues behind the scenes.

In addition to the Rationale instalments, this issue continues to draw from the Industrial session at the Ada-Europe conference 2005 in York. The two short articles we publish this time are from Per Sandberg and Rei Stråhle of Saab Systems, and from Jean-Pierre Rosen from Adalog. We have more such articles in store for the new year. We wish the organizers of the 2006 conference to be at least as successful with the level of participation and contributions, so that it can continue to richly feed the journal.

Before closing this editorial, I wish to gratefully acknowledge the precious contribution of the News editor, Santiago Urueña, the Calendar section and Forthcoming events editor, Dirk Craeynest (who wears many other hats as well, too many to mention ☺) and of the team in charge of printing and dispatching, headed by Michael Gonzalez-Harbour. It has been a pleasure to work with them and it will continue to be so in the future.

At this point it is in order to me to extend my very best personal wishes for New Year to our whole readership, our fellow members of Ada-Europe, its Board and its sibling national organisations.

Tullio Vardanega
Padova
December 2005
Email: tullio.vardanega@math.unipd.it

News

Santiago Urueña

Technical University of Madrid. Email: suruena@datsi.fi.upm.es

Contents

Ada-related Organizations

ACM SIGAda — SIGAda Awards Nominations

From: John McCormick <mccormick@cs.uni.edu>
Date: 5 Oct 2005 07:30:13
Subject: Approaching Deadline for Nominations for SIGAda Awards
Newsgroups: comp.lang.ada

Dear Members of the Ada Community:

On Wednesday, 16 November 2005, the 2005 SIGAda Awards will be presented in a special morning plenary session at the SIGAda 2005 conference in Atlanta, Georgia. (See http://www.acm.org/sigada/conf/sigada2005 if you have somehow missed announcements of this year's annual SIGAda international conference.)

We welcome your nominations of deserving recipients.

The ACM SIGAda Awards recognize individuals and organizations who have made outstanding contributions to the Ada community and to SIGAda. The two categories of awards are:

(1) Outstanding Ada Community Contribution Award — For broad, lasting contributions to Ada technology & usage.

(2) ACM SIGAda Distinguished Service Award — For exceptional contributions to SIGAda activities & products.

Please consider who should be nominated this year. You may nominate a person for either or both awards, and as many people as you think worthy. One or more awards will be made in both categories.

Please visit http://www.acm.org/sigada/exec/awards/awards.html#Recipients and peruse the names of past winners. This may help you think about the measure of accomplishment that is appropriate. You may be aware of people who have made substantial contributions that have not yet been acknowledged. Nominate them. Consider what you believe to be the best developments in the Ada community or SIGAda in the last year; the last 5 years; since Ada's inception. Who was responsible? Nominate them.

Please note that anyone who has received either of the two awards remains eligible for the other. Perhaps there is an outstanding SIGAda volunteer who has won our Distinguished Service Award and who has also made important contributions to the advance of Ada technology, or visa versa. Nominate him or her!

The nomination form is available on the SIGAda website at http://www.acm.org/sigada/exec/awards/awards.html. (You need to visit this website to see past award winners' names, and also a picture of the statuette which is the award among other things, so you don't nominate someone who has already won an award in a category.) Submit your nomination as an e-mail attachment to SIGAda-Award@acm.org. You may also submit nominations on-line at: http://www.acm.org/sigada/cgi-bin/ICRS-Register.cgi?Awards

The ACM SIGAda Awards Committee, comprised of volunteers who have previously won an award, will determine this year's recipients from your nominations.

Call our attention to the people who are most deserving, by nominating them. And please nominate by OCTOBER 15!

Your participation in the nominations process will help maintain the prestige and honor of these awards.

Thank you,

John McCormick

Chair ACM SIGAda

AdaIC — Ada Usage Survey

http://www.adaic.com/news/survey-results.html

Ada Market Entails at Least a $5.6 Billion Investment, Say Ada Resource Association-Sponsored Survey Respondees

BELMONT, MASS. [Sept. 12, 2005]— The Ada market is healthy, with a total investment of at least $5.6 billion in Europe and North America, according to an industry survey sponsored by the Ada Resource Association (ARA). The ARA, an international nonprofit organization, maintains the Ada Information Clearinghouse and comprises principal suppliers of Ada development environments and tools: AdaCore, IBM Rational Software, Polyspace Technologies, Praxis Critical Systems, and SofCheck.

The survey asked about both current Ada usage and familiarity with or plans for Ada 2005. Altogether, 188 responses were returned on the ARA website during May and early June this year, and additional data were derived from individual interviews. The survey was completed by software developers from North America, Australia, Korea, and almost every country in Europe.

The main results on Ada usage, presented at the Ada Europe conference in York, England, in June, can be summarized as follows:

* Around 322 million lines of Ada code (LOAC) are in software that is either still in development or has been completed, representing a reported (and conservatively-estimated) value of around $5.6 billion.

* The prices for these systems also cover a wide range. At one extreme, several projects undertaken by volunteers or hobbyists showed zero as their cost. And at the other end of the spectrum, a response for one of the major system developments reported a cost of $2 billion.

* The projects represent a variety of applications and stages of development. (In the table below, the percentages add up to more than 100% since some respondents checked off more than one category, such as "fielded" and "maintenance"):

Project type		
Embedded systems:	44	21%
Command & Control:	32	17%
Other Types:	32	17%
Tools:	30	16%
Simulation Projects:	30	16%
Graphics:	21	11%
Libraries:	11	6%
IT Projects:	7	4%

Project Stage		
Planning:	8	4%
Development:	78	41%
Complete:	31	16%
Fielded:	54	29%
Maintenance:	56	30%
Other stage:	13	7%

Outside the Defense/Aerospace Box

Although Ada's traditional stronghold has been in the defense/aerospace industry, the responses to the survey show that the language has a much broader appeal. This is likely due to Ada's intrinsic merit in helping produce reliable software, and to the availability of quality Ada compilers and tools. Some of the more interesting application areas include:

* Accounting
* Banking & Finances
* Bible Studies
* Book Title Image Matching
* Commercial Imaging
* Court Workflow
* Currency Trading
* Database Tools
* DNA Analysis
* Electronic Voting Machine
* Industrial Control
* Interlingual Machine Translator
* Internet Security
* Medical Devices & Testing
* Neuroscience Research
* Photonic Materials Research
* Security Assessment
* Semiconductor Factory
* Small Office Applications
* Spellcheck
* Telecommunications
* Tension Structure Analysis
* Warehouse Management/Control

Understanding and Using the New Ada Standard Features

Besides asking about ongoing projects, the survey collected data about respondents' acquaintance with, and usage plans for, features that are being added to the upcoming Ada standard. The survey offered six possible answers for each feature, from "Unaware" and "Do not understand" to "Frequently use." The following specific features were listed: "limited with"; interfaces; scheduling improvements; the container library; nested extensions; prefixed views; directories/environment/ calendar packages; enhanced anonymous access types; limited aggregates and functions; overriding indicators; Ravenscar; expanded Unicode support. A roughly one line description was given of each.

Since tutorial or rationale material on Ada 2005 has only recently been made available to the general Ada community, and since most of the information available has been instead very technical, a high degree of familiarity with the new features would have been somewhat surprising. The actual results — on average, about 34% of the respondents said that they either were unaware of a feature, didn't understand it, or didn't answer the question — is probably better than expected and reflects a high degree of interest in the new language

New Ada Standard Features Likely to be Used

The best understood new features were the containers library and the other new packages, while the least understood feature was overriding indicators. The description in the question didn't explain their use, unlike most of the other features, which might have explained respondents' confusion.

Those features that respondents said they would never use proved to be highly specialized. Further, if a feature was understood, it would tend to be used: on average, more than 80% of the users who understood a feature said that they would use it at least occasionally. An interesting counterexample was the Ravenscar Profile: 32% of the respondents that understood that feature said that they would never use it. This may seem surprising, since the Ravenscar profile is generally regarded as one of Ada's major strengths for high-integrity applications. But most of the survey's respondents are working on systems that, although requiring high reliability, are not safety-critical. The developers can thus use the full Ada language rather than a specialized subset.

The feature most likely to be used, by developers who indicated an understanding of the feature, is the new standard packages (for directories /environment variables/calendar), followed by the containers library, prefixed views, and overriding indicators.

In reviewing the survey data, Randy Brukardt, editor of the new ISO Ada standard and long-time member of the Ada Rapporteur Group (ARG), observed: "All-in-all, I would say these results validate the ARG's effort in choosing how to update the language. The survey responses showed a higher degree of familiarity with the features than we had expected, and it is especially interesting that the new libraries are perceived as the most useful addition. With the forthcoming Rationale and other articles, the Ada community will be able to learn more about what the new standard will offer."

For those who did not get an opportunity to fill it out, the survey is still available. Updates from new surveys and individual interviews will be posted in the future in a news article and to the Ada News group, which you can sign up for by emailing listserv@adaic.com and putting "Subscribe announce" in the subject line. For more details, please see http://www.adaic.org/site/newslist.html

AdaWorks Closing

From: <adaworks@sbcglobal.net>
Date: Sat, 22 Oct 2005 18:45:00 GMT
Subject: AdaWorks Closing
Newsgroups: comp.lang.ada

We started AdaWorks in 1987. Since then we have had a good time doing consulting, programming, and training in the world of Ada. From time to time we have branched out into other areas, but we have stayed close to our core business, that of helping to make Ada an attractive alternative for software developers world-wide.

I recently terminated our web site. It was generating more spam than anything else. Existing clients who want still want my individual services know how to reach me by telephone. They also have my private email account, not published on any of the public sites.

During the last five years, most of my time has been devoted to teaching software engineering topics at the Naval Postgraduate School, where I continue to promote Ada whenever I can. In two weeks, I will start the Ada module of a comparative programming class I teach twice a year.

I am now on the threshold of my eighth decade. The days of being away from home for up to two weeks out of every month, which is what my Ada consulting required, is not fun anymore. I still enjoy business-related traveling, but need to cut back — a lot.

I have closed the AdaWorks web site. The name, AdaWorks, will persist for a while in other contexts. I will continue to visit and contribute to this forum. I plan to update Ada Distilled to the 2005 standard sometime next Spring. My email address, richard@adaworks.com will no longer work after today.

For those of you who know me, there is no cause for worry. I am still in robust health, continue to practice Judo two or three evenings a week, love my teaching duties at NPS, and hope to see some of you from time to time at various conferences and other meetings. I am still an active member of GAP for NPS.

Richard Riehle

From: Adrian Hoe <abyhoe@gmail.com>
Date: 28 Oct 2005 19:47:29
Subject: Re: AdaWorks Closing
Newsgroups: comp.lang.ada

I join the others to wish you and your family all the best. We have never met but if you recall, we have exchanged some ideas. I wish to meet you someday.

I'm using your Ada Distilled for reference and to teach Ada. I love it! It is an excellent book.

Ada-related Events

[To give an idea about the many Ada-related events organized by local groups, some information is included here. If you are organizing such an event feel free to inform us as soon as possible. If you attended one please consider writing a small report for the Ada User Journal. --su]

Feb 25–26 — FOSDEM 2006

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 28 Nov 2005 10:43:27 +0100
Subject: Ada at FOSDEM 2006
Newsgroups: comp.lang.ada

The Free and Open-Source Developers' Meeting (FOSDEM) is an annual event held in Brussels, Belgium, in February. The 2006 edition will take place on Saturday the 25th and Sunday the 26th of February, 2006. Ada-Belgium has organised a series of presentations related to Ada, to be held in a dedicated developers' room, all day Sunday. Here is the programme:

10:00h–11:00h Jean-Pierre Rosen: Introduction to Ada

Jean-Pierre will put his well-known talent to good use, introducing Ada to beginning or experienced programmers alike.

11:00h–12:00h Jean-Pierre Rosen: AdaControl

AdaControl is a tool that analyses Ada source text and verifies compliance with coding rules and guidelines. AdaControl is free software written under contract with Eurocontrol, and takes advantage of ASIS, the standard interface that allows Ada programs to analyse Ada source text. Jean-Pierre will introduce AdaControl, ASIS, and the business model that allows one to make a living writing free software.

12:00h–13:00h Philippe Waroquiers: Use of free software in European

Air Traffic Flow Management

Philippe Waroquiers leads software development of the ETFMS system at Eurocontrol, the European air traffic control agency with 34 member states. Software on which millions of travellers' lives each year depend is written in Ada using AdaCore's Free Software Ada compiler, GNAT Pro.

13:00h–14:00h lunch break

14:00h–15:00h Ludovic Brenta: Ada in Debian

Ludovic Brenta will explain his work as the main maintainer of Ada in Debian, and the policy that unites all Ada packages, thereby making Debian the best free Ada development platform in the world :) This will be an excellent opportunity for a tour of existing free software projects developed in Ada.

15:00h–16:00h AdaCore: Ada Academic Initiative

AdaCore is the company that offers technical support and consulting services around GNAT, the GNU project's free Ada compiler. AdaCore is also the main developer of GNAT. The Ada Academic Initiative aims to encourage universities and other education institutions worldwide to use and teach Ada, by offering a broad range of services at no cost to professors and students. If possible, AdaCore will demonstrate the latest GNAT Programming Studio with GNAT GPL 2005 Edition.

16:00h–17:00h AdaCore: The PolyORB schizophrenic middleware

An example of fruitful collaboration between academia and industry, PolyORB allows heterogeneous software components to communicate with one another by bridging various middleware technologies such as CORBA, MOM and the Ada Distributed Systems Annex (annex E).

All presentations will be in English, but all speakers also speak French. You may ask questions on comp.lang.ada, fr.comp.lang.ada, or join the AdaFOSDEM mailing list (in English).

More information:

FOSDEM : http://www.fosdem.org

AdaCore : http://www.adacore.com

Free Software from AdaCore: http://libre.adacore.com (includes, among others, GNAT, GPS and PolyORB which will be the focus of some talks)

Debian : http://www.debian.org

Eurocontrol : http://www.eurocontrol.int

Ada-Belgium : http://www.cs.kuleuven.ac.be/~dirk/ada-belgium/

AdaFOSDEM mailing list, operated by Ada-Belgium: http://listserv.cc.kuleuven.ac.be/archives/adafosdem.html

[See also "Feb 26-27 - Ada event at FOSDEM 2005" in AUJ 25-4 (Dec 2004), p.181 --su]

Jun 5–9 — Ada-Europe 2006

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Organization: Ada-Europe, c/o Dept. of Computer Science, K.U.Leuven
Date: 16 Nov 2005 00:18:14 +0100
Subject: C.f.Industrial Pres., Reliable Software Technologies, Ada-Europe 2006
Summary: Eight weeks until submission deadline!
Keywords: Conference,tutorials,reliable software,Ada,LNCS,Portugal
Newsgroups: comp.lang.ada,fr.comp.lang.ada

Call for Industrial Presentations

11th International Conference on Reliable Software Technologies - Ada-Europe 2006

5 - 9 June 2006, Porto, Portugal

http://www.ada-europe.org/conference2006.html

Organised, on behalf of Ada-Europe, by Instituto Superior de Engenharia do Porto in cooperation with ACM SIGAda (approval pending)

General Information

The 11th International Conference on Reliable Software Technologies (Ada-Europe 2006) will take place in Porto, Portugal. Following the usual style, the conference will span a full week, including a three-day technical program and vendor exhibitions from Tuesday to Thursday, along with parallel workshops and tutorials on Monday and Friday.

Call for Presentations

In addition to the usual call for papers, and considering the success achieved in the previous conference, we are having a call for presentations primarily aimed at industrialists who have valuable experience to report but who do not wish to write a complete paper.

This separate call for presentations is made for Experience Reports from Industrial Projects and/or Experiments, Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics and Experience Reports on Education and Training Activities, with bearing on any of the conference topics.

See below for further details.

Schedule

12 January 2006: Submission of presentation proposals

20 January 2006: Notification to authors

28 April 2006: Presentation material required

5-9 June 2006: Conference

Submission of Presentations

Presenters are invited to submit a one-page overview of the proposed presentation to Peter Dencker (peter.dencker@aonix.de) by January 12th 2006. The Industrial Committee will review the proposals.

The authors of selected presentations shall prepare their final presentation, together with a short abstract (max 10 lines), by 28th April 2006; they should aim at a 20 minutes talk. The authors of accepted presentations will also be invited to derive articles from them, for publication in the Ada User Journal.

Exhibitions

Commercial exhibitions will span the three days of the main conference. Vendors and providers of software products and services should contact the Exhibition Chair José Ruiz as soon as possible for further information and for allowing suitable planning of the exhibition space and time.

Conference Topics

In the last decade the conference has established itself as an international forum for providers and practitioners of, and researchers into, reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers from industry, academia and government organizations interested in furthering the development of reliable software technologies. To mark the completion of the technical work for the Ada language standard revision process, contributions that present and discuss the potential of the revised language are particularly sought after.

For papers, tutorials, and workshop proposals, the topics of interest include, but are not limited to:

- Methods and Techniques for Software Development and Maintenance: Requirements Engineering, Object-Oriented Technologies, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues

- Software Architectures: Patterns for Software Design and Composition, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-Based Design

- Enabling Technology: CASE Tools, Software Development Environments and Project Browsers, Compilers, Debuggers and Run-time Systems

- Software Quality: Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems

- Critical Systems: Real-Time, Distribution, Fault Tolerance, Information Technology, Safety, Security

- Mainstream and Emerging Applications: Multimedia and Communications, Manufacturing, Robotics, Avionics, Space, Health Care, Transportation

- Ada Language and Technology: Programming Techniques, Object-Oriented Programming, Concurrent Programming, Distributed Programming, Bindings and Libraries, Evaluation & Comparative Assessments, Critical Review of Language Enhancements, Novel Support Technology, HW/SW platforms

- Experience Reports: Experience Reports, Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics, Experience Reports on Education and Training Activities with bearing on any of the conference topics

Tutorials

Tutorials should address subjects that fall within the thrust of the conference and may be proposed as either half- or full-day events. Proposals should include a title, an abstract, a description of the topic, a detailed outline of the presentation, a description of the presenter's lecturing expertise in general and with the proposed topic in particular, the proposed duration (half day or full day), the intended level of the tutorial (introductory, intermediate, or advanced), the recommended audience experience and background, and a statement of the reasons for attending. Proposals should be submitted by e-mail to the Tutorial Chair Jorge Real. The providers of full-day tutorials will receive a complimentary conference registration as well as a fee for every paying participant in excess of 5; for half-day tutorials, these benefits will accordingly be halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorial in issues preceding and/or following the conference.

Workshops

Workshops on themes within the conference scope may be arranged to discuss matters of immediate technical interest as well as to foster action on longer-term technical objectives. Proposals may be submitted for half- or full-day workshops, to be scheduled on either ends of the main conference. Workshop proposals should be submitted by e-mail to the Conference Chair Luís Miguel Pinho. The workshop organiser shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Organizing Committee

Conference Chair

Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal, lpinho@dei.isep.ipp.pt

Industrial Committee Co-Chairs

Peter Dencker, Aonix GmbH, Germany, peter.dencker@aonix.de

Michael González Harbour, Universidad de Cantabria, Spain, mgh@unican.es

Industrial Committee (preliminary list)

Rod Chapman, Praxis High Integrity Systems
Chris Smith, GreenHills
Franco Gasperoni, AdaCore
Jacques Brygier, Aonix
Ian Gilchrist, IPL
Pascal Leroy, IBM Rational
Rei Stråhle, Saab Systems
Francis Thom, Artisan Software
Tony Elliston, TNI Europe
Luís Miguel Pinho, Conference Chair
Dirk Craeynest, Ada-Europe (Vice President)
Erhard Plödereder, Ada-Europe (President)

Conference Organization

Conference Chair

Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal, lpinho@dei.isep.ipp.pt

Program Co-Chairs

Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal, lpinho@dei.isep.ipp.pt

Michael González Harbour, Universidad de Cantabria, Spain, mgh@unican.es

Tutorial Chair

Jorge Real, U. P. Valencia, Spain, jorge@disca.upv.es

Exhibition Chair

José Ruiz, AdaCore, France, ruiz@adacore.com

Publicity Chair

Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium, Dirk.Craeynest@cs.kuleuven.be

Local Chair

Sandra Almeida, Polytechnic Institute of Porto, Portugal, salmeida@dei.isep.ipp.pt

Ada-Europe Conference Liaison

Laurent Pautet, Telecom Paris, France, pautet@enst.fr

Ada and Education

SPARK Training

http://www.praxis-his.com/sparkada/training.asp

Public Course Dates for 2006 - UK

Course 1 – "Software Engineering with SPARK"

6th - 9th March 2006 at the Praxis High Integrity Offices in Bath.

Course 2 – "Black-Belt SPARK"

21st - 23rd March 2006 at the Praxis High Integrity Offices in Bath.

Course 3 – "High-Integrity Concurrent Software Design with RavenSPARK"

10th March 2006 at the Praxis High Integrity Offices in Bath.

Course 4 – "UML to SPARK"

10th March 2006 at the Praxis High Integrity Offices in Bath.

[See also "SPARK Training" in AUJ 26-2 (Jun 2005), p.72 --su]

Ada-related Resources

Contributors for the Ada Programming Wikibook

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Wed, 14 Sep 2005 20:11:50 +0200
Subject: "Ada Programming" need some help form ObjectAda and APEX users.
Newsgroups: comp.lang.ada

We the Authors of "Ada Programming" have a little problem: We have only access to the GNAT compiler. It would be nice also to provide project files for other Ada compilers.

And for that we need the help from Ada users which have access to the other Ada tool chains.

If you have some time and access to a non GNAT compiler do go to

http://en.wikibooks.org/wiki/Ada_Programming/Building

and help us out.

See also "Ada Wikibook is 'Book of the month'" in AUJ 26-3 (Sep 2005) p.151 and "Ada at Wikipedia & Wikibooks" in AUJ 26-1 (Mar 2005) p.8 --su]

Updated Ada Resources

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Wed, 05 Oct 2005 08:54:39 +0200
Subject: Re: Investigating Ada
Newsgroups: comp.lang.ada

> I have been looking into Ada as a new language to pick up. I tend to use C for most projects, but I enjoy learning new things. I'm already sold on the value of the language itself from a technical point of view, but my concern is that it might have a rather small userbase, especially without the Ada Mandate. I am having trouble finding recent FAQs or usage information. It's easy to come across something from like 1994, but that's over ten years ago ;)

Ahh yes, the old pages have a better Google rating - no one knows why. Here some pages with newer dates:

September 29, 2005, at 08:12 AM:

http://ada.krischik.com/

15:05, 1 October 2005 203.197.76.86:

http://en.wikibooks.org/wiki/Ada_Programming

Tuesday October 4th, 2005

http://www.adaworld.com/

From: Jeff Creem <jcreem@yahoo.com>
Subject: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada
Date: Thu, 08 Sep 2005 18:31:08 -0400

Oh well. It was time for my annual (or at least bi-annual) trip to the whois database to watch to see if the ada home dot com domain was going to expire (not written as a URL to avoid giving googlebot more links to the site than it already has).

Ada home dot com was once a great site for up to date Ada information but about 6 years ago it stopped getting updated at all (other than the automated headline script at the left side of the page that shows the date).

Unfortunately it is still highly ranked in the search engines and its horribly stale content (some of which is still relevant of course) lends to the perception of a dead language.

Well… The reason it is a sad day is that today when I checked it had been renewed again but this time until 2010..

Created: September 12, 1996

Modified: September 8, 2005

Expires: September 10, 2010

So consider this a reminder for those that have Ada websites that any links to Ada home dot com should be removed to reduce the page rank.

I am not sure we will ever get it off the first page but it would feel good to see "Ada programming" or "Ada language" drop this page down a few pegs.

Be sure to add links to:

http://www.adapower.com - General Ada and Ada programming information

http://www.adaworld.com - General Ada and Ada programming information

http://en.wikipedia.org/wiki/Ada_programming_language

and of course the award winning

http://en.wikibooks.org/wiki/Ada_Programming

From: David Trudgett <wpower@zeta.org.au>
Date: Fri, 09 Sep 2005 10:43:08 +1000
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

About six years ago [Magnus Kempe claimed] to be happy to have a competent volunteer maintain his FAQ (why does an individual "own" the FAQ for Ada?)

Has Magnus disappeared off the face of the earth or something? Is it true that in six years no one has been interested enough to create another FAQ which is up-to-date and not controlled by an individual?

From: Jeff Creem <jcreem@yahoo.com>
Date: Thu, 08 Sep 2005 22:26:19 -0400
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

Not sure what came of that but both David and I have tried to contact Magnus several times over the past few years about transfer of the entire site/domain and he was not interested. I never saw or heard of an offer to have someone else maintain the site until that repost (which was just for the FAQ as near as I could tell)

And the reason the FAQ is "owned" by a person is it is copyrighted and therefore can not be copied to another site.

(See the bottom of the page: This FAQ is Copyright 1994-1996 by Magnus Kempe.) Of course in the US even without an explicit statement it would have likely been protected by copyright.

It does have fairly liberal terms for re-distribution but the part that says "as long as it is completely unmodified" sort of causes some issues. Of course most of the FAQ and answers really originally came from other people so technically the individual items are actually not copyright'd by Magnus but by the original authors.

The fun part of his response is where he claims David is mistaken that Magnus no longer wants to actively work on the website… but then the site is never updated once after that…

In any case, it is (was) of course a volunteer effort and he is under no obligation to do anything… so to some extent I was needlessly venting.

From: David Trudgett <wpower@zeta.org.au>
Date: Fri, 09 Sep 2005 18:08:00 +1000
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

I knew about the copyright issue, but my "why" was asking a deeper question. What thought processes would lead a person to take other people's contributions from comp.lang.ada, incorporate them into a FAQ, and then claim exclusive ownership over it? It seems like an inappropriate, selfish and controlling act to me. But I do not know the circumstances, hence the question. The corollary (another aspect of my original question) is, "Why was he allowed to do it?" (i.e., I didn't notice any controversy about it in c.l.a archives, though it's possible I could have missed it).

Maybe he feels he is preserving an important piece of history… but why insist on camping on the domain name?

The real problem, though, doesn't seem to be Magnus Kempe or adahome, but rather the fact that Google lists it right up the top of the search results (when *I* tried it, at least). As you know, this must have to do with how many sites link to it. Why do they link to it if it's out of date? Perhaps it still has content that can't be found elsewhere? If so, why is that after six years?

…and it's too bad about the American Dental Association, too! ;-)

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Fri, 09 Sep 2005 20:39:14 GMT
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

Strictly speaking, that statement only says that he's never *expressed* that he isn't working on the website. It doesn't really say that he's actually working on it, and it certainly doesn't say anything about what he *wants* to do.

I bet he's been wanting to get back to Ada Home all this time, and he's probably constantly thinking he'll soon find some time for it. I can understand that situation. Myself I have many unfinished projects that I've never consciously abandoned. They've just slipped away while other things kept me busy. (None of them are as public as Ada Home though.) Don't you have some of those too?

The only thing about Magnus Kempe that seems strange to me is that he doesn't want to let anyone else update Ada Home, if that is indeed the case.

> As you know, this must have to do with how many sites link to it. Why do they link to it if it's out of date?

Because they're outdated too?

From: Tom Moran <tmoran@acm.org>
Date: Fri, 09 Sep 2005 01:15:27
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

He apparently isn't the only FAQ-non-maintainer: look at www.faqs.org/faqs/computer

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sat, 10 Sep 2005 19:32:41 +0200
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

That's one of the reasons I changed all my web-sites to wiki technology - so anybody can make a fix if I have not time. Currently WikiSpam is not a real problem so I don't even have a password in place.

From: Larry Kilgallen <Kilgallen@SpamCop.net>
Date: 8 Sep 2005 20:01:10 -0500
Subject: Re: Sad Day for Ada - Update your links
Newsgroups: comp.lang.ada

Why post a general request like that rather than going to the 167 people involved?

http://www.google.com/search?as_lq=www.adahome.com

How to Compile GNAT 2005

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sun, 25 Sep 2005 10:26:51 +0200
Subject: GNAT/GPL DIY update.
Newsgroups: comp.lang.ada

I finally managed to compile GNAT/GPL myself. Had to copy two files from the GPS source tree into the GCC tree.

Current status as always on:

http://ada.krischik.com/index.php/Articles/CompileGNATGPL

It's a wiki so do add your own experiences to it. Ask for the upload password I you have specialised build batches to contribute.

[See also "How to compile GCC 3.4" in AUJ 25-3 (Sep 2004), p.120 --su]

Ada-related Tools

Implementation of new Ada 2005 packages

From: Afraid of Software <info@afraidof.co.uk>
Date: 17 Oct 2005 15:04:40 -0700
Subject: [ANN] Ada.Environment_Variables
Newsgroups: comp.lang.ada

Following the recent discussion on the new standard package Ada.Environment_Variables, we have decided to release our implementations of this package for Windows-based compilers.

Ada 95 - For GNAT and ObjectAda

Ada 200Y - For GNAT only

They are released under the terms of the GNAT-Modified GPL and come with GPS and ObjectAda project files, ready to build.

Please see http://www.afraidof.co.uk/ to download.

SPARK and the GNAT Programming Studio (GPS)

From: Zheng Wang <zw@cs.man.ac.uk>
Date: 30 Nov 2005 08:24:36 -0800
Subject: Spark in Gnat Programming System (GPS)
Newsgroups: comp.lang.ada

In the 2005 version of GPS, SPARK tools has been ported to GPS. I found it is very useful and helpful to program Ada/SPARK in GPS, because it has sophisticated IDE and is a really good compilation and debugging environment. Also it has free versions for both software developer and academia. So, I would like to recommend SPARK users in academic to have a try on GPS.

[See also "GNAT GPL 2005 Edition" in AUJ 26-3 (Sep 2005), p.153–154 --su]

Archive of old GNAT public versions

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Organization: Ada-Belgium, c/o Dept. of Computer Science, K.U.Leuven
Date: 12 Oct 2005 22:02:38 +0200
Subject: Re: 3.15p is disappearing! (but not everywhere…)
Summary: Still available on the Ada-Belgium archive.
Newsgroups: comp.lang.ada

Jean-Pierre Rosen wrote:

> I just checked: 3.15p is no more on cs.nyu.edu, nor on its mirrors (at least the french one).

The GNAT distribution on cs.nyu.edu was removed between Tuesday August 09 at 05:30 GMT+2 and Wednesday August 10 at 05:30 GMT+2.

I noticed, because for over a decade each day at that time the automatic mirror script for the Ada-Belgium ftp archive runs. After some checking what was going on, I have ensured that the mirror script no longer tries to update our GNAT mirror.

The full GNAT 3.15p distribution, as was on cs.nyu.edu up to Aug 09, is and will remain available on the Ada-Belgium ftp archive at <ftp://ftp.cs.kuleuven.be/pub/Ada-Belgium/mirrors/gnu-ada>.

In addition, the "OLD" directory contains other binary distributions that were at some time on cs.nyu.edu (e.g. for Solaris, HP-UX, AIX, etc.), as well as the compiler source distributions from 3.10p on.

If there are other compiler/tools distributions that should be mirrored feel free to contact me and I'll see what we can do.

[See also "ACT - New Public Release GNAT 3.15p" in AUJ 24-1 (Mar 2003), p.17 --su]

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Organization: Ada-Belgium, c/o Dept. of Computer Science, K.U.Leuven
Date: 17 Oct 2005 19:32:51 +0200
Subject: Re: 3.15p is disappearing!
Summary: … but mirrors remain …
Newsgroups: comp.lang.ada

Simon Clubley wrote:

> It's not just 3.15p that's gone. They appear to have pulled the kits for other operating systems, regardless of version, as well. For example, an old version, 3.12p, was available for VMS Alpha at: ftp://ftp.cs.nyu.edu/pub/gnat/private/old/openvms/ That directory no longer exists.

A copy is and will remain available on the Ada-Belgium ftp archive at

<ftp://ftp.cs.kuleuven.be/pub/Ada-Belgium/mirrors/gnu-ada/OLD/3.12p/openvms/>

About the GNAT GPL 2005 License

From: Olivier Issaly <kiad@free.fr>
Subject: Re: GNAT GPL 2005 Edition is now available
Newsgroups: fr.comp.lang.ada
Date: Fri, 21 Oct 2005 10:43:20 +0200

Ludovic Brenta wrote [translated from French – su]:

> This post has degenerated into an interesting discussion on the next Ada compilation system for Debian. I calling on those interested to post their viewpoints on comp.lang.ada.

A little summary would help those who arrived late to this thread and do not dare read through 200 posts …

That's the summary that I can make out after reading the top layers:

* AdaCore distribute their GNAT version only under the GNU General Public License (GPL), while previously the licence in force was GNAT Modified GPL (GMGPL), which permitted the creation of non-libre software using GNAT.

* The consequence is then that those software products that use GNAT should be placed under GPL, which no longer permits the use of other licences.

* One still could use the official GCC distributions, which continue to include the special clause from GMGPL.

* AdaCore work primarily on the development version of GCC and does not feed back to the older stable versions of GCC other than security patches. Consequently, the GMGPL version that we had has suddently stopped being maintained.

* One can of course continue to work with the CVS version of GCC at her own risk though.

* Those who most suffer from this string of events are the SME that do not develop libre software but do not have the financial means to pay what AdaCore charge for GNAT Pro.

* This policy change on the part of AdaCore has spawned lots of misunderstandings (has there been any official statement from them in this respect yet ?).

[See also "GNAT GPL 2005 Edition" in AUJ 26-3 (Sep 2005), p.153–154 --su]

From: Marc A. Criley <mc@mckae.com>
Date: Tue, 04 Oct 2005 15:15:03 -0500
Subject: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

I wrote to AdaCore to ask them what their maintenance and upgrade intentions were for the GNAT GPL Edition.

I received a quite timely response that said their intent is to "[keep] this version roughly in sync with the latest GNAT Pro releases" and "in fact the latest GPL edition is more recent than the latest GNAT Pro official release, it corresponds to the more recent GNAT Pro Ada 2005 Beta".

Maybe Ludovic should rerun his Debian survey? Declining to include a maintained Ada development environment on a Linux distribution may be unwise…

[See also "Next Debian Ada Compiler" in this issue --su]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 5 Oct 2005 04:10:19 -0700
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

> I think that this [change of license of GNAT's run-time system from GMGPL to GPL] goes *against* the spirit of free software. As far as I can understand, the basis of free software is "you can do anything with this software, *except* deny to others the rights you have received". Anything, including proprietary software.

With the GMGPL, you receive the right to see and modify the source code of the GNAT run-time; but you can also deny others this same right. The GPL is more "free" than the GMGPL, since with it you cannot deny others this right anymore. This is the "free" spirit as defined by [Richard M.] Stallman and the [Free Software Foundation (FSF)]. The BSD license does allow you to deny rights to others, and has a different definition of "free".

> The GPL edition is a big mistake made by AdaCore, both from a marketing and a popularity point of view. If the community clearly refuses this edition (and not having it in Debian is quite a strong indication), maybe AdaCore will revise its policy.

This is a possibility.

From: Steve <steved94@comcast.net>
Date: Thu, 13 Oct 2005 18:41:01 -0700
Subject: GNAT GPL Edition - on the plus side
Newsgroups: comp.lang.ada

My experience:

GNAT GPL Edition includes the first version of GPS I have run on Windows that I haven't removed after a few days of frustration.

The compiler seems to just plain work (as usual).

I do find that the pretty-printer gets confused with Ada 2005 code.

IMHO the folks at AdaCore have done a great job.

[See also "AdaCore — GPS 3.0" in AUJ 26.2 (Jun 2005), p.77 --su]

From: Hyman Rosen <hyman.rosen@gmail.com>
Date: 6 Oct 2005 12:20:15 -0700
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

Jeffrey R. Carter wrote:

> But if I create an application using my GMGPL code and the GNAT GPL compiler, my program (and all of its source code) must become GPL, and I am not allowed to [let others add my GMGPL code to a propietary application].

That's not true. When you distribute your sources along with your binary, there is nothing stopping you from adding that special exemption to the license for your own code. Then the people to whom you have distributed can build an executable using your code and distribute it to others without giving them the source to your code, just as you would like them to be able to do. It just means that they can't do it using the GNAT GPL compiler.

> Thus, I find the GNAT GPL compiler unsuitable for my purposes. I consider it inappropriate for any compiler to take such choices from me, and for any OS to supply such a compiler.

It's unsuitable for people who want to build programs not bound by the GPL. But it's fine if they want to build such programs and grant additional exemptions on their own code.

From: Simon Wright <simon@pushface.org>
Date: Fri, 07 Oct 2005 06:54:35 +0100
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

Jeffrey R. Carter wrote:

> The GPL is quite clear that a program that uses GPL code in any way falls under the GPL. If the run-time library is GPL code, then any program that uses the run-time library is GPL.

No, it is not GPL, it must be released under the terms of the GPL. I suppose there might be an argument that a person who distributes a binary that mixes GPL code and the Booch Components (which are GMGPL) would have to make available the BC sources under GPL terms, but the recipient could always come back to me and ask for a fresh copy under GMGPL. Or indeed any other terms (they would fail, probably, because it's not just my copyright in there, the other authors would have to agree too). I just think this is all FUD.

From the Libre site:

«Q. I would like to release my software under the XYZ license, which is a Free Software license according to the FSF, but is incompatible with the GPL. What should I do?

A. The GNAT GPL Edition doesn't limit in any way the license you use on your sources. If you are distributing sources only, no issue with respect to the license of GNAT GPL Edition arises. You or anyone who wants to build a binary can do so freely from these sources, using either the GNAT GPL compiler or any other suitable Ada compiler. If you want to *distribute* a binary of your program compiled with the compiler in the GNAT GPL Edition then *today* the binary must be licensed under the GPL. Note that you can still license a copy of your sources under the XYZ Free Software license of your choosing. It is AdaCore's intention to work with the FSF to modify the licensing of the GNAT GPL Edition to allow the use of other Free Software licenses for binaries produced with the compiler inside the GNAT GPL Edition. Meanwhile, you can distribute in source form only.»

From: Marc A. Criley <mc@mckae.com>
Date: Fri, 14 Oct 2005 11:10:39 -0500
Subject: Re: Licences
Newsgroups: comp.lang.ada

The GPL doesn't stop you from selling your product. And for as much money as you want to ask and can get.

The GPL doesn't forbid you from requesting that your customer not redistribute your product (though all you can do is make the request, you can't legally stop them from doing so, all you can do is let them know you will refuse to support them or provide updates if they do).

If your customer doesn't actually link your product into their code, i.e. you're building an application rather than a library or such, that customer is not required to GPL their code.

If your product does need to be linked in, then it depends on whether the customer must use GPL GNAT to utilize it, or can they build and link using a differently licensed compiler/runtime? If it's "vanilla" Ada, even GNAT Ada, then you can develop your code with GPL GNAT and release it under any license you want. (The key thing is whether someone could utilize your product with a GNAT compiler other than GPL GNAT, if so, you're in the clear.)

The GPL does allow your customer to redistribute your product, though as mentioned above you can request they not do so. (Ref: GNAT GAP and GNAT Pro.)

The GPL does require that you provide access to your product's source code. (But if you were going to use the GMGPL you were going to do that anyway, right?)

For help on understanding what the GPL makes you do, and also allows you NOT to do, see the GPL FAQ at http://www.gnu.org/licenses/gpl-faq.html.

IANAL

From: Jeff Creem <jcreem@yahoo.com>
Subject: Re: Licences
Date: Sat, 15 Oct 2005 10:12:49 -0400
Newsgroups: comp.lang.ada

Lucretia wrote:

> Why do people keep pretending that AdaCore's removal of the GMGPL license exception from their runtime isn't a really big change for the Ada community? It really does make a difference to a lot of individual programmers (and thus to Ada). Programmers who cannot afford to pay AdaCore (or who choose not to) now must rely on significantly out of date versions of compilers and tools in order to "make money" on a program they wrote.

No. Not exactly. Programmers who cannot afford to pay AdaCore (or who choose not to) must either rely on significantly out of date versions of the compilers. Or they must rely on versions from the FSF tree. Or they must rely on non-free but still relatively cheap offerings from RRSoftware (or perhaps Aonix)

> I believe this will produce a code fork in the FSF tree and people like me will continue to use that version of GNAT, unless another free compiler becomes available.

I am not sure why a fork is needed. AdaCore appears to still be contributing to the FSF tree and there has been no discussion at all in the FSF GCC groups about a license change. AdaCore goes through quite a bit of work to package and test and "put their name" on a particular GNAT version and they have decided (right or wrong..for them and us) that it is best for their business if they cause some limiting of the use of their freely distributed versions.

The FSF tree does exist. It does continue to be maintained and it is (probably) in the best interest of AdaCore to keep working within that tree to some extent.

I find it interesting when people talk of wanting a $0 dollar Aonix build when a big reason given for why the FSF tree is not acceptable is the lack of the distributed systems annex and ASIS. Are those things available from Aonix? (I honestly do not know).

I am not trying to say that I am happy with the situation. I just don't think that the FSF tree approach is really all that bad. There are of course quality issues with the FSF tree at times but more and more non-ada GCC developers appear to at least be enabling Ada during their bootstraps.

The quality will probably never get where we'd like it as long as the state of Ada in the tree is not a consideration for the release criteria. I am not sure if this will ever change but if a few people stepped up to the plate and became GCC developers supporting the Ada tree it would not hurt.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Fri, 14 Oct 2005 18:59:31 +0200
Subject: Re: Licences
Newsgroups: comp.lang.ada

> So, I can continue to use FSF GNAT (I think). Is GCC 4.x going to get the GPL treatment as well?

I am aware of no plans to change the license in the FSF tree. In any case, AdaCore cannot make such a change by themselves; only the FSF can do that. AdaCore made a license change only on their own internal sources, by applying a clause of the GPL.

> Also, using the current GCC, if I distribute a binary, do I have to link it to a shared libada (or whatever it's called)?

You don't have to use libgnat, you can use pragma No_Run_Time, but then most of the benefits of using Ada vanish. Whether you link statically or dynamically makes no difference from a licensing point of view; it is a purely technical decision. In any case, if you use the GMGPL libgnat, you must provide the sources of libgnat, or point your customers to gcc.gnu.org where they can obtain the sources for themselves. You do not have to provide the sources for your program.

If you use a libgnat under GPL, then you must provide the sources for you program, too. But you can still sell your program (and services surrounding your program: hint, hint) for money.

From: Brian May <bam@snoopy.apana.org.au>
Date: Sat, 15 Oct 2005 20:18:22 +1000
Subject: Re: Licences
Newsgroups: comp.lang.ada

Steve Whalen wrote:

> I know of at least one programmer who wrote and sold a program after I got him interested in Ada via the GNAT compilers. That cannot happen anymore because a binary only program cannot be "sold" from the AdaCore GPL compiler (the program that was sold that I'm referring to here did NOT generate enough money to pay for an AdaCore support contract, though it did pay for a few presents for the wife to make up for late nights spent on the computer). It is now much

> harder to get these kinds of Ada converts without a free Ada compiler.

It would appear AdaCore has forgotten the "proprietary programs written by individuals or small companies that cannot afford and do not want to pay for an expensive support contract" market. As well as open source programs that do not use a GPL compatible license (for any number of reasons - personally like others I like using current versions of GPL in my own code).

I could understand this if Ada was more popular.

In a way, the goal of the license is similar to the free version BitKeeper - free for open source but paid (at high rates geared at big projects IRC) for closed source software. At one time I believe the code was "open source" (but restrictive license). Then it changed to free binary + restrictive license. I think the situation has now changed, I don't think they support a free version anymore - lets hope AdaCore doesn't revoke the free version in the same manner (I got the impression at LCA2005 that this was a bitter dispute with various parties throwing insults and accusations at each other - I think on the linux-kernel mailing lists).

Fortunately, I believe in this case, FSF holds the copyright, so presumably AdaCore can't restrict access any more then making it GPL.

However, I think the people AdaCore "forgot" will use the GCC GNAT compiler now instead. This in turn hopefully will lead to my development with this compiler.

I suspect the end result is that people will end up continuing to use the compiler they are most familiar with - even when the project justifies paying the $$$ for the AdaCore supported compiler.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Organization: Tidorum Ltd
Date: Fri, 28 Oct 2005 13:20:35 +0300
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

> I'm curious. Do you actually know of any commercial software written in Ada? That is, a program written in Ada which is sold as a non-customized, shrink-wrapped product to multiple customers? Do any of them show up here in c.l.a? Are they complaining?

I am not the original poster, but in part answer to your question, I am writing software in Ada that is meant to be sold mainly as a non-customized product to multiple customers.

I did "complain" about GNAT GPL 2005 in the sense that I voted for some other GNAT (that is, a GMGPL GNAT) to be the Debian Ada compiler. But I see that GNAT GPL 2005 has many advantages, perhaps most importantly less work for the Debian Ada team. I am prepared to use some of my own effort and money to have a non-GPL compiler, for example the FSF GNAT. I am also experimenting with the Janus/Ada compiler. The price of the Aonix compiler would be a significant hurdle for me and GNAT Pro is beyond my ceiling at this time.

So, that's one data-point for you. As I recall, some of the others who voted for a GMGPL Debian compiler were in the same position.

Fuzzy sets for Ada

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 12 Sep 2005 22:30:12 +0200
Subject: ANN: Fuzzy sets for Ada v4.1
Newsgroups: comp.lang.ada

It is here:

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

Changes to version 4.0:

1. Multiplication and division between a fuzzy number from one side and a dimensioned number or interval from another yielding a dimensioned fuzzy number were added;

2. Multiplication and division between a linguistic variable and a dimensioned number yielding a dimensioned variable were added;

3. Multiplication and division of linguistic variables were defined to work with negative multiplicands and divisors;

4. To_Variable_Measure with Variable argument was added;

5. Empirical defuzzification methods of linguistic variables: center of area, center of gravity, leftmost maximum, rightmost maximum;

6. Bug fix in implementation of operations on linguistic variables.

[See also same topic in AUJ 26-2 (Jun 2005), p.75 --su]

Multiprecision Numbers

From: jtg <jtg77@poczta.onet.pl>
Date: Thu, 15 Sep 2005 19:06:52 +0200
Subject: What about big integers in Ada 2005?
Newsgroups: comp.lang.ada

When I started to learn Ada, one of the most interesting features for me was the possibility to declare integer type of the specified range. I imagined that the integer type may be of any size. However, several years later I needed big integers and I was disappointed - Ada 95 does not support integers of ANY size, it can support only those integers that are supported by the processor, or smaller.

Does Ada 2005 support big integers? Where can I learn how to use them?

From: gdemont@hotmail.com
Date: 16 Sep 2005 07:33:26 -0700
Subject: Re: What about big integers in Ada 2005?
Newsgroups: comp.lang.ada

Big integers are special beasts that have to be treated differently than "normal" integers because they can have an arbitrary size in memory and require a specific handling not foreseen in "normal" processors.

However big integers are doable and available in every Ada version; on the following site you'll find a bunch of downloadable packages:

http://www.chez.com/bignumber

BTW, the freshest version of my multi-precision integers are in mathpaqs.zip on my Ada page below.

From: Craig Carey <research@ijs.co.nz>
Newsgroups: comp.lang.ada
Subject: Re: What about big integers in Ada 2005?
Date: 16 Sep 2005 20:43:15 -0700

That web page is out of date since not referring to my Ada bindings to the GNU GMP big numbers package:

http://tope.tigris.org/source/browse/tope/src/Gmp/gmp_2k4.ads?view=markup

My code replaces the GMP Ada bindings named at the chez.com website.

GMP did compile well with GCC 2.8.1 (which was needed since combining GNAT 2.8.1 with a GCC C (not C++) of a different version, in Windows, led to botched parameter passing.

There was a message about C/C++ to Ada. A way to get that running is to have two Ada YACC (AdaGOOP) parsers in series. Maybe Mr de Montmollin can produce a program to convert C body files into faulty Ada.

[See also same topic in 25.2 --su]

QtAda binding

From: Pascal Obry <pascal@obry.net>
Date: Mon, 14 Nov 2005 08:13:21 +0100
Subject: [ANNOUNCE] QtAda binding from Leonid Dulman
Newsgroups: comp.lang.ada

Leonid Dulman (developer of VAD, the Visual Ada Developer) ask me to post this as he does not have write access to comp.lang.ada at the moment. He has done a QtAda binding, the work can be found there:
http://www.websamba.com/guibuilder

This has been tested with Qt 3.3.

From: Preben Randhol <randhol@bacchus.pvv.ntnu.no>
Organization: Norwegian university of science and technology
Date: Mon, 14 Nov 2005 13:09:07
Subject: Re: [ANNOUNCE] QtAda binding from Leonid Dulman

Newsgroups: comp.lang.ada

Great. Pity it is GPL and not GMGPL, but still it is nice. Seems to be a thin binding?

Cheddar - Real-Time Scheduling Simulator

From: Singhoff <singhoff@fraise.univ-brest.fr>
Organization: Universite de Bretagne Occidentale
Date: 8 Nov 2005 11:06:48
Subject: Ann. : new release of Cheddar, a free real time scheduling analyzer
Newsgroups: comp.lang.ada

The LYSIC team is pleased to announce a new release of Cheddar.

Cheddar is a free real time scheduling tool. Cheddar is designed for checking task temporal constraints and buffer sizes of a real time application/system. It can also help you for quick prototyping of real time schedulers. Finally, it can be used for educational purposes.

Cheddar is developed and maintained by the LYSIC Team, University of Brest.

Cheddar is composed of two independent parts: an editor used to describe a real time application/system, and a framework. The editor allows you to describe systems composed of several processors which own tasks, shared resources, buffers and which exchange messages. The framework includes many feasibility tests and simulation tools. Feasibility tests can be applied to check that task response times are met and that buffers have bounded size. When feasibility tests can not be applied, the studied application can be analyzed with scheduling and buffer simulations. Cheddar provides a way to quickly define "user-defined schedulers" to model scheduling of ad-hoc applications/systems (ex: ARINC 653).

Cheddar is written in Ada. The graphical editor is made with GtkAda. Cheddar runs on Solaris, Linux and win32 boxes and should run on every GNAT/GtkAda supported platforms

The current release is now 1.3p5. If you are a regular Cheddar's user, we strongly advice you to switch to the 1.3p5 release due to the large amount of 1.3p3 bugs that we fixed.

Cheddar is distributed under the GNU GPL license. It's a free software, and you are welcome to redistribute it under certain conditions; See the GNU General Public License for details. Source code, binaries and documentations can be freely downloaded from http://beru.univ-brest.fr/~singhoff/cheddar

1) Summary of features:

- Do scheduling simulations with classical real time schedulers (Rate Monotonic, Deadline Monotonic, Least Laxity First, Earliest Deadline First, POSIX queueing policies: SCHED_OTHERS, SCHED_FIFO and SCHED_RR) with different type of tasks (aperiodic, periodic, task activated with a Poisson process law,…)

- Extract information from scheduling simulation (buffer utilization factor, task response times, task missed deadlines, number of preemption,…)

- Apply feasibility tests on tasks and buffers (without scheduling simulation):

  + Compute task response time bounds.

  + Apply processor utilization tests.

  + Compute bound on buffer size (when buffers are shared by periodic tasks)

  + Shared resources support (scheduling and blocking time analysis). Supported protocols: PIP, PCP.

- Tools to express and do simulations/feasibility tests with task precedences:

  + Schedule tasks according to task precedences

  + Compute Tindell end to end response time.

  + Apply Chetto and Blazewicz algorithms.

- Tools to run scheduling simulation in the case of multiprocessors systems

- Do simulation when tasks are randomly activated.

- Can run scheduling simulation on user-defined scheduler and task arrival patterns.

- Run user-defined analysis on scheduling simulation.

- …

2) Most of new features provided by 1.3p5:

- Fix many bugs of the previous release (see BUGS file)

- Add AADL import and export. This feature is based on the Ocarina AADL parser distributed by the ENST (F. Singhoff)

- Perform several optimization in order to increase global performance. (ex: decrease memory footprint)

- Add round robin and time sharing built-in schedulers (F. Singhoff)

- Add FreeBSD Makefile.vars (C.P. Gloster)

- Feasibility tests can be called by users processor by processor and feasibility test by feasibility test. (see customized feasibility tests) (F. Singhoff)

- Refactoring of some basic widgets (H. Lapinoja, F. Singhoff)

- Improve text displayed in the main root window when response time and feasibility tests are performed on a project (F. Singhoff)

- Extend the data model of cheddar to include the abstraction of address spaces (F. Singhoff)

- Some parametric variable can be set into the parametric code (F. Singhoff)

- Provide extended buffer analysis tools: simulation and feasibility tests in cases of periodic and randomly activated tasks (J. Legrand, F. Singhoff)

- Users can tune which events the simulator engine will generate at simulation time (see Tools/Scheduling/Options) Warning: "task_activation" and "running_task" can not be ungenerated since they are needed by many analysis tools (F. Singhoff)

Thanks to H. Lapinoja, J. Legrand, T. Vergnaud, T. Ren, K. Bryan, C.P. Gloster, J. Stemerdink and F. R. De la Rocha for their help on this new release (tests, bug reports or new features)

3) Work in progress:

For the next year, we plan to improve the tool with the following features:

- Provide a global memory analysis (task stack, text/data segments,…)

- Add Hierarchical schedulers support.

- Updated the graphical editor in order to take into account GtkAda 2.2

- Update the user's guide according to the new provided features

- Improvement of message scheduling with:

  + End to end response time with message scheduling.

  + Allowing message sending at any time of a task capacity

  + Providing a way to user-defined message delay communication by specification of user-defined message scheduling (as user-defined scheduler)

- Fix a buggy service which should detect deadlock from simulation.

[See also same topic in AUJ 25-4 (Dec 2005), p.191–192 --su]

Ada Support in Eclipse

From: Jeff Creem <jcreem@yahoo.com>
Date: Wed, 16 Nov 2005 22:18:55 -0500
Subject: Eclipse Ada Support - FYI
Newsgroups: comp.lang.ada

While poking around I see that AdaCore has announced (perhaps not yet released) support for a GNAT Pro Eclipse plug-in (primarily targeting wind river's workbench which is Eclipse based).

http://www.gnat.com/pressroom_27.php

There are now three Ada vendors with Eclipse support in some fashion (DDC-I, Aonix and AdaCore).

Still none that are publicly available.

There were some discussions a few months back on the Eclipse CDT mailing lists that some on the the CDT group were interested in supporting Ada but I have not heard a lot since then.

From: Jeff Creem <jcreem@yahoo.com>
Subject: Re: Eclipse Ada Support - FYI
Date: Sat, 26 Nov 2005 17:20:02 -0500
Newsgroups: comp.lang.ada

Alex Xela wrote:

> I would like to comment on one assertion: "Still none that are publically available". It is not true as the Aonix plug-in has been available now for 2 months, is available on Linux and Windows, and is used on actual projects.
> It is free when used with ObjectAda and one version for GNAT is also available. Even if the plug-in is enhanced every day, the current versions are of industrial quality and perfectly usable for real Ada developments.

Yes. I did not choose my words carefully enough. I meant publically available in terms to imply that all people on this list could just go and download it for free. (or simply type in a credit card and pay for it for that matter).

ObjectAda is like many other products targeted at the giants. If you are a home user, even if you have the money to spend, you can't figure out how to spend it (or what the cost is) without dealing with some sales staff. Not really a knock on Aonix. It is pretty standard for all things targeting "the enterpruse" that even the people who make the product recommendations can't get a reasonable story on cost, availability, licensing, etc without talking to someone.

Of course I could be missing the point of your e-mail. If there is a URL for a version that can either be downloaded for free or for which I can type in a credit card and buy it, it would be nice to know as this is one of the better kept secrets in the world of Ada as far as I can tell.

[See also "Ada Plugin for Eclipse" in AUJ 26-3 (Sep 2005), pp.154–155 --su]

AdaControl

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 22 Sep 2005 13:54:20 +0200
Subject: AdaControl V1.4 released
Newsgroups: comp.lang.ada
Organization: Adalog

Adalog is pleased to announce a new release of AdaControl, the free rules checker for Ada.

Main new features:

Interactive mode, with new commands to check rules interactively and/or try command files.

Better support of attributes

New rule type: count, allowing AdaControl to be used as a measuring tool.

Improvements to the framework

Small improvements, bug fixes, etc.

New rules:

Declarations
Exception_Propagation
Naming_Convention
Real_Operators
Representation_Clauses
Side_Effect_Parameters
Simplifiable_Expressions
Specification_Objects
Statements
When_Others_Null

AdaControl is available from Adalog's component page at http://www.adalog.fr/compo2.htm (for detailed information, the user guide and programmer manual are directly accessible too).

AdaControl is a professional product, extensively tested on Eurocontrol's software (thanks to P. Waroquiers). Adalog can provide support and maintenance for AdaControl.

AdaControl is fully GMGPL, which means that you can use it for any purpose and even reuse any part of it in any software. ASIS users: there are some general-purpose utilities that can be very helpful!

[See also same topic in AUJ 26-2 (Jun 2005), p.73 --su]

From: Jean-Pierre Rosen <rosen@adalog.fr>
Organization: Adalog
Date: Tue, 04 Oct 2005 10:29:27 +0200
Subject: New release of AdaControl 1.4
Newsgroups: comp.lang.ada

This is a minor release (1.4r20) that just fixes a bug and improves performance for the rule Local_Hiding.

No change in functionality.

Scout — Ada utility for Google Earth & NASA World Wind

From: Tom Moran <tmoran@acm.org>
Date: Tue, 29 Nov 2005 17:27:38 -0600
Subject: Ann: AdaWorld posting
Newsgroups: comp.lang.ada

Stephane Richard has kindly posted to www.adaworld.com an application, "Scout" that reads/writes/converts latitude&longitude (preserving precision), Metes&Bounds ("thence N 12d 34' 56" E 78.9 ft" style), State Plane Coordinates, and path files for Google Earth and NASA WorldWind. Includes both Ada source code and Windows executable.

[See also same topic in AUJ 26-3 (Sep 2005), p.155 --su]

From: Marc A. Criley <mc@mckae.com>
Date: Wed, 30 Nov 2005 08:44:14 -0600
Subject: Re: Ann: AdaWorld posting
Newsgroups: comp.lang.ada

Tom was also good enough to incorporate a feature whereby Scout will accept as input the Tab Separated Value (TSV) output from the Garmin GPS MapSource application. One simply needs to cut out everything but the Trackpoint lines from the file, either before or after loading it into Scout, and can then convert those points into whatever formats it supports.

This has worked great for me when taking my GPS along on an ATV ride through the hills around my place, exporting the track log, running it through Scout to get a Google Earth kml file that I can overlay onto the terrain and do flyovers--impressed the heck out of my neighbor :-)

Ada rocks! :-)

High Seas Battleship - Network Game

From: David Trudgett <wpower@zeta.org.au>
Date: Tue, 08 Nov 2005 21:19:26 +1100
Subject: New Ada ANSI console/X terminal game
Newsgroups: comp.lang.ada

For those who may be interested, I've just posted to my website a first release (version 0.99) of my "High Seas Battleship" network game. You can find the source code for it at:

http://www.zeta.org.au/~wpower/dkt/programs/high-seas-battleship.tar.gz

It's a full-screen colour text-based application written for Linux console or xterm, but should work on most ANSI compatible terminal devices. My program makes use of Samuel Tardieu's AdaSockets, which I doubt works on Windows (the manual seems to mention only Unix and OpenVMS), so I likewise doubt that High Seas will work as it is under Windows, even assuming that there is a suitable ANSI terminal available for Windows. It should be easy, however, to adapt the comms layer to use a different sockets library, if desired.

For non-GNAT users, I should probably mention that the program uses a GNAT specific library (in the main procedure, battleship.adb) to access environment variables, so you'll probably want to replace this with the relevant vendor-specific library, or perhaps Ada.Command_Line.Environment (I didn't use it myself because GNAT warned me that this is an internal GNAT package that therefore shouldn't be used).

Besides AdaSockets and GNAT.OS_Lib just referred to, there are no other dependencies except that you'll probably want a C compiler to compile a couple of

utility routines (which could probably be omitted, though).

To compile under Linux/Unix, assuming you have GNAT and libadasockets installed, you will probably just need to edit the Makefile and run make.

The project was pursued simply as an Ada learning exercise, so comments would be appreciated, if you have any.

From: Pascal Obry <pascal@obry.net>
Date: Tue, 08 Nov 2005 18:38:14 +0100
Subject: Re: New Ada ANSI console/X terminal game
Newsgroups: comp.lang.ada

No problem. AdaSockets does work on Windows we have been using it for AWS.

From: David Trudgett <wpower@zeta.org.au>
Date: Sat, 12 Nov 2005 20:05:47 +1100
Subject: Re: New Ada ANSI console/X terminal game
Newsgroups: comp.lang.ada

Those who have downloaded the initial release may like to know that I have posted a small bugfix to my High Seas Battleship game. The updated version is at:

http://www.zeta.org.au/~wpower/dkt/programs/high-seas-battleship_v0.99a.tar.gz

This fixes a bug whereby it was possible to change the position of one's fleet mid-game by typing the caret symbol. Oops… ☺

Ada-related Products

AdaCore — GNAT Pro AltiVec Support

AdaCore Brings Ada to AltiVec Support For Leading Weapons Company

New York, October 3, 2005

AdaCore, the leader in Ada solutions, today announced GNAT Pro Ada development tool suite support for Freescale Semiconductor, Inc.'s high-performance AltiVec™ instruction set. The ability to access AltiVec instructions from Ada was initiated by MBDA, a world leading, global missile systems company. MBDA wanted to combine the potential power of AltiVec with the 'formality' of Ada. By adopting Ada, the programming language best suited for safety-critical, high-reliability applications, MBDA is supporting its own stated priority to "deliver the highest product reliability while developing innovative customized solutions."

"Using Ada with AltiVec gives MBDA the ability to reliably extract the maximum performance from the smallest number of processors," said Rod White, Technologist, MBDA, at the recent Ada-Europe conference in June 2005. "The result is lower system costs and reduced power consumption, plus Ada's rigorous structuring ensures long-term code maintainability."

MBDA was attracted to Ada as a "strongly typed" programming language, a characteristic that is key to application reliability and security. AltiVec access from Ada allows MBDA to concentrate on operations, letting Ada datatypes solve data alignment issues, and achieving very low overheads. MBDA's choice of AdaCore to bring Ada to AltiVec is consistent with the company's strategy "to pursue and reinforce the development of strong, mutually beneficial links with industry leaders worldwide."

"Missile systems clearly require a high-integrity, safety-critical programming language," said AdaCore president, Robert Dewar. "Ada's uncompromising emphasis on sound software engineering, structure, and formality lets MBDA exploit the advanced performance of AltiVec while still ensuring the absolute reliability and integrity of the weapons systems."

About AltiVec

AltiVec technology expands the capabilities of PowerPC microprocessors by providing excellent general-purpose processing performance, while concurrently addressing high-bandwidth data processing and algorithmic-intensive computations. Using Ada to access the SIMD (Single Instruction Multiple Data – sometimes referred to as vector processing) AltiVec instruction set allows programmers to capitalize on the parallel processing features increasingly common on modern processors. AltiVec's adds 162 "vector" instructions to versions of the PowerPC including Motorola's G4 and IBM's G5 processors.

AdaCore's newly released integrated development environment (IDE), called the GNAT Programming Studio (GPS), will be included as part of the AdaCore solution for AltiVec.

Pricing and Availability

AdaCore's solution for AltiVec is available today.

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial, open-source software solutions for Ada, a modern programming language designed for large, long-lived applications where reliability, efficiency and safety are absolutely critical. AdaCore's flagship product is GNAT Pro, the commercial-grade open-source Ada development environment, which comes with expert online support and is available on more platforms than any other Ada technology. AdaCore has customers worldwide; see http://www.adacore.com/customers.php for more information.

Use of Ada and GNAT Pro continues to grow in high-integrity and safety-critical applications, including commercial and defense aircraft avionics, air traffic control, railroad systems, financial services and medical devices. AdaCore has North American headquarters in New York and European headquarters in Paris. www.adacore.com

AdaCore — Support for VxWorks 6

AdaCore Announces Support for Wind River's VxWorks 6

New York, November 7, 2005

AdaCore, a Wind River leading Ada technology supplier, today announced that its industry-leading GNAT Pro Ada development environment will support version 6 of Wind River's VxWorks® RTOS. Developed in conjunction with Wind River, AdaCore's GNAT Pro for VxWorks 6 is targeted to the Wind River® General Purpose Platform on the PowerPC, from Windows, GNU/Linux and Solaris host environments.

GNAT Pro is a well-established and widely used product on Wind River platforms, with several hundred customers supplied with GNAT Pro for VxWorks to date. GNAT Pro for VxWorks 6 will offer many new enhancements, including memory protection models and plug-in support for the Wind River Workbench development suite to provide a fully-integrated Ada solution.

"With such a large body of customers already using GNAT Pro with VxWorks, AdaCore clearly has the field-proven experience to deliver one of the most solid Ada implementations to real-time systems developers," said Rob Hoffman, Senior Director, Aerospace and Defense at Wind River Systems. "The new version is perfect for safety-critical and real-time embedded situations where reliability, performance and portability are absolutely required."

"As a complete Ada implementation with real-time performance and functionality, GNAT Pro continues to be the natural Ada solution for the evolving capabilities of VxWorks," said Robert Dewar, President of AdaCore. "This new version marks the continuation of a successful and longstanding partnership between AdaCore and Wind River, and the most meaningful resulting benefit is the high quality and performance we jointly bring to embedded software developers."

GNAT Pro for VxWorks 6 includes the following features:

* Implementation of all versions of Ada: Ada 2005, Ada 95, Ada 83

* Support for VxWorks 6 Kernel Modules and Real-Time Processes

* Mixed-language support, allowing composition of applications comprising Ada, C, and C++

* Full source-level debugger

* Extensive GNAT library

* Ada unit testing framework (AUnit)

* Full source code for GNAT Pro

* Dependable support from AdaCore

AdaCore — GNAT Pro Support for Wind River Workbench Development Suite

AdaCore Brings the Power of GNAT Pro to Wind River's Workbench Environment

New York, November 7, 2005

AdaCore, a Wind River leading Ada technology supplier, today announced a GNAT Pro plug-in developed specifically for the Wind River® Workbench development suite. Especially suited for large, embedded, real-time applications running VxWorks® RTOS, AdaCore's GNATbench solution combines the reliability of Ada development and compilation technology with Wind River's popular Eclipse-based, open device software development framework. The result is a fully integrated Ada toolset that enhances Workbench to facilitate multi-language development, sophisticated editing, browsing, debugging, and comprehensive compilation for advanced VxWorks systems creation.

GNATbench draws its strength from AdaCore's GNAT Pro, a robust and flexible Ada development environment based on GNU GCC compiler technology. GNATbench extends the Workbench text editor to support sophisticated Ada-aware editing and browsing, adds code generation for building systems in Ada or in combination with any of the other languages supported by the Wind River Workbench development suite, and supports Ada-aware debugging with the Wind River Workbench debugger. GNATbench utilizes the GNAT Pro compiler and tool-suite for the VxWorks platform, the same comprehensive Ada compiler that Wind River has offered to customers for many years. GNATbench is supported on Windows, Linux and Solaris and is planned for release Q1, 2006.

"Wind River and AdaCore have collaborated for more than seven years to bring Ada's safety critical features and VxWorks RTOS capabilities to hundreds of customers worldwide," said Rob McCammon, Director, Product Management at Wind River Systems. "Our continued success demonstrates the value that dedicated partnerships bring to the development of faster, more reliable, and cost-effective device software."

"With one simple plug-in, GNATbench lets Workbench users access the benefits of GNAT Pro Ada," said Robert Dewar, President of AdaCore. "Our new tools seamlessly integrate into the Wind River Workbench development suite to facilitate ease of use, and deliver the power of GNAT Pro's leading-edge compiler and tools to provide more reliable applications with fast, predictable performance."

[See also "SCORE Support for Wind River Workbench Development Suite" in this issue --su]

Aonix — ObjectAda 8.2 for Linux

Aonix® ObjectAda 8.2 Boasts More Language Support and Dynamic Debug Capabilities

Linux® support delivered to meet customer demand

http://www.aonix.com/pr_09.12.05c.html

Embedded Systems Conference, Boston, MA, September 12, 2005

Aonix®, a provider of solutions for safety- and mission-critical applications, released the latest version of the ObjectAda development environment, targeting Linux as the first supported operating system. In addition to supporting an Eclipse-based development environment targeting mission-critical software solutions, ObjectAda 8.2 integrates a number of product enhancements. With a newly developed capability to attach the symbolic debugger to a running Ada application, ObjectAda has become especially useful for situations in which programming errors do not surface until after the test and debug phase of development has been completed.

ObjectAda 8.2 is the first ObjectAda release where developers can choose between the traditional Aonix IDE for development and the new AonixADT™ Eclipse plug-in. AonixADT incorporates Ada-project awareness, an Ada-language sensitive editor, Ada-language compile and build capabilities, and a complete Ada debugger interface, enabling Ada developers to enjoy state-of-the-art interface capabilities geared to maximize developer ease and efficiency. By delivering the powerful combination of Eclipse and best-of-class Ada technology, Aonix provides a modern and robust development environment that increases developer productivity. Since AonixADT is based on Eclipse, this same integrated development environment also supports development with the Java, C, and C++ languages.

"Aonix has a long-standing reputation of supplying high-quality development tools based on industry accepted standards," said Jacques Brygier, VP Marketing of Aonix. "Increasingly, our Ada development customers require large-team access to broad ranges of integrated development tool technologies. By basing AonixADT on the popular open-source Eclipse development environment, Aonix maximizes the ease with which the various best-of-breed technologies can be integrated.

ObjectAda® for Linux comes with both a graphical and command-line interface, integrated language-sensitive editor, lightweight source-based library model, and industry leading compilation speed. The ObjectAda for Linux compilation system is composed of the editor, source-code browser, compiler, debugger, and full library manager.

In addition to the basic compiler development package, the ObjectAda Project Pack contains the Ada-ASSURED advanced editor that provides additional language-sensitive features and style-guideline conformance checking. ObjectAda Project Pack also contains the AdaNav™ toolset, which provides complete system HTML source-navigation capabilities as well as call- and unit-tree graphical reporting and automatic data dictionary generation. To improve program performance, the AdaNav profiler provides run-time performance reporting to identify application hot spots.

The ObjectAda 8.2 family has many other products currently in development. Scheduled for release over the next several months are ObjectAda for Windows®, ObjectAda Windows for Intel®x86/ETS™, ObjectAda Linux for PPC/LynxOS®, ObjectAda Solaris™ for PPC/LynxOS, ObjectAda Windows for PPC/RAVEN™, ObjectAda Solaris for PPC/RAVEN and ObjectAda Windows for PPC/VxWorks®.

Shipping and Availability

ObjectAda for Linux is available immediately for Red Hat Enterprise Version 4 and Fedora Core Version 4. For more information about this product, please visit www.aonix.com/objectada.html

About Aonix

Aonix offers mission- and safety-critical solutions primarily to the military and aerospace, telecommunications and transportation-related industries. Aonix delivers the leading high-reliability, real-time embedded virtual machine solution for running Java™ programs deployed today and has the largest number of certified Ada applications at the highest level of criticality. Our unique modeling solution features UML™ 2.0 profiles and MDA™ tailored for the mission- and safety-critical space. Aonix products include PERC®, RAVEN™, and Ameos™. Headquartered in San Diego, CA and Paris, France, Aonix operates sales offices throughout North America

and Europe in addition to offering a network of international distributors. For more information, visit www.aonix.com.

Aonix — ObjectAda 8.2 for Windows

ObjectAda 8.2 for Windows Delivers .NET Compatibility and Dramatic Performance Improvements

Addresses emerging requirements for coordination with Java and .NET components

Birmingham, UK, October 19, 2005

http://www.aonix.com/pr_10.19.05.html

Aonix®, a provider of solutions for safety- and mission-critical applications, released the latest version of ObjectAda for the Windows development environment. In addition to supporting an Eclipse-based development environment targeting mission-critical software solutions, ObjectAda for Windows integrates current Microsoft platform improvements, ensuring full compatibility with Microsoft Visual Studio .NET, faster linking times, as well as other platform enhancements. In preparation for market trends, ObjectAda for Windows includes a Java-call interface, enabling Java applications to be called from an Ada program.

The Aonix ObjectAda for Windows brings the improvements of ObjectAda 8.2 to the Windows development platform. In integrating current Windows improvements with the Aonix Ada 95 compiler, Aonix has delivered enhancements to the object code and symbolic debugging information generation and provided full compatibility with the Microsoft Visual Studio .NET 2003 development tools. Recognizing the growing number of large-scale Ada projects, ObjectAda 8.2 for Windows offers dramatic performance improvements for developers linking executable files or initiating debugging sessions for large programs.

“The increased number of large-team projects requires simplified, platform access,”” noted Jacques Brygier, vice president of marketing at Aonix. “By providing a Windows- and Eclipse-compatible technology for Aonix customers, we ensure that they have access to the broadest range of integrated development tool technologies, reducing cost and improving project flexibility. Our ObjectAda for Windows enables developers to take advantage of Eclipse-based resources as well as enjoy seamless integration within the Microsoft environment.”

ObjectAda for Windows 8.2 includes the comprehensive Ada libraries needed for calling Windows Win32 and the Visual C++ .NET 2003 MFC interfaces from application source code written in Ada. In ObjectAda for Windows, these Ada binding libraries are fully compatible with the Microsoft Visual Studio .NET 2003 tools and libraries.

In addition to the basic compiler development package, ObjectAda Project Pack contains AdaJNI, an interface to call Java™ programs from Ada and the AdaNav™ toolset, which provides complete system HTML source-navigation capabilities as well as call- and unit-tree graphical reporting and automatic data dictionary generation. The AdaNav profiler also provides run-time performance reporting to identify application hot spots. The ObjectAda Pinnacle contains the Ada-ASSURED advanced editor that provides additional language-sensitive features and style-guideline conformance checking.

As part of the ObjectAda 8.2 family, ObjectAda for Windows allows developers to choose between the traditional Aonix IDE for development and the new AonixADT™ Eclipse plug-in. AonixADT incorporates Ada-project awareness, an Ada-language sensitive editor, Ada-language compile and build capabilities, and a complete Ada debugger interface, enabling Ada developers to enjoy state-of-the-art interface capabilities geared to maximize developer ease and efficiency.

Shipping and Availability

ObjectAda for Windows is available immediately for Windows 2000 and XP platforms. For more information about ObjectAda 8.2 for Windows, please visit: www.aonix.com/objectada.html.

DDC-I — SCORE Support for Wind River Workbench Development Suite

DDC-I Announces State-of-the-Art Ada Support for Eclipse Based Wind River® Workbench Development Suite

Proven code generation with all the features and middleware of the VxWorks® RTOS

October 3, 2005 – DDC-I, a global leader in safety critical software tools for embedded applications today announced a new plug-in for the Wind River Eclipse-based Workbench development suite. This new plug-in provides multi-language support (Ada, C and Embedded C++), targeting the VxWorks RTOS on Intel 80x86 (including VxSim) and PowerPC 603 & 604 core processors.

DDC-I’s SCORE® development tools (including compiler, linker, disassembler, library management and object dumper), are seamlessly integrated into the Wind River Workbench development suite and do not open another unique window. Through the project management view of the Workbench tools, the user has the ability to compile, link and build both downloadable kernel modules, and real-time processes with mixed language code including Ada 95, C, Embedded C++, and Assembler. DDC-I’s build process for Ada files automatically determines a correct compilation order when building or compiling multiple files. SCORE® is also integrated with the Wind River debugger and enables source-level debugging of Ada 95 code, C code, and Embedded C++ code, switching seamlessly between the languages.

"This is a true integration," states Bob Morris, President and CEO at DDC-I, Inc. "No more launching one tool from another; the entire edit, build, and debug activities are all controlled from within Workbench - the industry leading, Wind River development suite."

"Industry partners like DDC-I help provide quality software tools needed for device software applications," states Rob Hoffman, Senior Director Aerospace & Defense at Wind River Systems. "We are pleased that this seamless integration is now available for our safety critical customers and look forward to working with them in the future."

SCORE® adds the following features to the Wind River Workbench development suite:

Hierarchical setting of compilation options through property editors.

Context sensitive actions to compile, link, build and disassemble Ada 95 source code.

A view for detailed compiler output, also showing exactly what compilation options are set on each file as it is being compiled.

Automated building of Ada 95 and mixed language projects - in the background.

Selective Linking to eliminate unused code portions from each object file.

Powerful error reporting back to the Workbench tools.

About DDC-I, Inc.

DDC-I, Inc. is a global supplier of software development tools, custom software development services, and legacy software system modernization. DDC-I's customer base is an impressive "who's who" in the commercial, military, aerospace, and safety-critical industries. Tools include compiler systems and run-time systems for C, Embedded C++, Ada, JOVIAL and Fortran application development. For more information regarding DDC-I products, contact DDC-I at: 400 North Fifth Street, Phoenix, Arizona 85004; phone (602) 275-7172; fax (602) 252-6054; e-mail sales@ddci.com; or visit www.ddci.com.

[See also "AdaCore — GNAT Pro Support for Wind River Workbench Development Suite" in this issue --su]

DDC-I — Migration Assessment Package to Ada

Customer Loyalty Brings New Opportunities for DDC-I

Phoenix, Arizona -- November 1, 2005 -- DDC-I today announced that the company's initiative to help companies migrate legacy code to new hardware and programming languages is winning support from some of the industry leading defense contractors. Numerous existing customers are asking for assistance in tossing out the old and bringing in the new. Last August, the company announced project-specific Migration Assessment Packages for programs facing this daunting task. Today, customers making this move are seeing successful results for many reasons.

Development tools on outdated hardware can be migrated to newer technology and can drastically improve productivity and future flexibility. For example, SCORE® from DDC-I can instantly offer mixed language, multi-target capability, allowing for easier integration of older programming languages with newer languages, such as Ada83 and C++.

"Our customers know the value of our tool suites and services, and they appreciate how our #1 in Customer Care policy produces customer service which is unmatched in the industry" stated Bob Morris, President and CEO, DDC-I, Inc.

DDC-I's Migration Assessment Package begins with on-site migration needs assessment and application/data/infrastructure evaluation culminating in a complete migration assessment report. Once the assessment is complete, the customer can make an informed choice of how to move forward. They can also choose to do the work themselves, or let DDC-I handle it for them with significant time and costs savings.

Green Hills Software and I-Logix — Unified Integrated Development Environment

Green Hills Software and I-Logix Redefine "Integrated Development Environment" with First Single Source Solution Addressing All Phases of Embedded Systems Development

Complete Standards-Based Approach from UML 2.0 for Design through POSIX for Deployment with Eclipse for Enterprise-Wide Integration

SANTA BARBARA, CA and ANDOVER, MA—October 3, 2005—Green Hills Software and I Logix announced today a new strategic partnership resulting in the first single source, fully standards-based Integrated Development Environment (IDE) that addresses all phases of embedded systems development, from requirements specification through deployment. Under the agreement, Green Hills Software will distribute and support I Logix Rhapsody, the award winning Model-Driven Development (MDD) environment based on the Unified Modeling Language (UML) 2.0 standard, together with Green Hills Software's MULTI and AdaMULTI development environments and the certified POSIX conformant INTEGRITY real-time operating system (RTOS). The products are seamlessly integrated to enable a unique bi-directional workflow between modeling and implementation. The companies are also collaborating on future integrated features and capabilities, including integration with the Eclipse platform.

Traditionally, system designers and developers have used separate environments for different aspects of development: one for requirements analysis through design, often paper-based, and another for implementation through deployment. This approach is very inefficient, since the design has to be re-created in the implementation and any changes to the design or implementation must be manually reflected in the other view. In contrast, the unified solution developed by Green Hills Software and I-Logix accelerates time-to-market by generating the implementation in C, C++ or Ada source code directly from the UML model. The model is then automatically updated to reflect any changes made to the code. The combination of Rhapsody and MULTI also helps ensure that a final product satisfies its design objectives by providing traceability between the source code and requirements.

"The Green Hills Software and I-Logix partnership is a harbinger of a dramatic change in the way embedded systems are developed," commented Dr. Jerry Krasner, Principal Analyst at Embedded Market Forecasters. "Our year-over-year research results consistently show that Model-Driven Development reduces development time and risk while also improving the alignment between pre-design expectations and final products. By providing a single-source, integrated solution for analysis, modeling, simulation, coding, debugging and deployment, Green Hills Software and I-Logix are not only bringing MDD into the mainstream by significantly improving its usability and accessibility, but are uniquely addressing the emerging requirements being set forth for new Aero/Defense, Automotive and Telecom markets in particular."

"As long-term users of Rhapsody, MULTI, and INTEGRITY, we are encouraged by this strategic partnership," said Bob Greene, Software Process Improvement Manager for L-3COM, makers of real-time embedded communications products for the US Government. "With this tightly integrated solution we now have a single cohesive development environment that enables us to analyze, model, design, implement and test our embedded software applications. The strengthening of the relationship and the continued enhancement to the integration of these tools should help to improve our workflow and the productivity of our staff."

"The I-Logix and Green Hills Software partnership is a powerful combination of market leaders, both of whom are the fastest growing suppliers in their respective market segments," commented Chris Lanfear, Practice Director for Embedded Software at Venture Development Corporation (VDC). "In addition to their complementary technology, both companies have momentum in fast growing markets including aerospace, defense and communications equipment."

Integrated Solution Addresses All Phases of Development

The combination of Rhapsody, MULTI and INTEGRITY provides an integrated solution that addresses all phases of embedded systems development.

* Requirements analysis and design—using UML with Rhapsody.

* Validation—Application behavior can be simulated in Rhapsody based on the UML model.

* Implementation—Rhapsody can automatically generate C, C++ and Ada source code for input to Green Hills C/C++ and Ada compilers. In addition, source code changes made in the MULTI editor are synchronized back in the Rhapsody model.

* Debugging and optimization—The MULTI source-level debugger is fully synchronized with the Rhapsody UML models. Execution breakpoints can be set and visualized at both the model and source-code levels. The application can also be simulated using the MULTI instruction set simulator if target hardware is unavailable.

* Testing—Rhapsody can automatically generate test vectors from the model that can be executed on the target hardware or MULTI simulator.

* Deployment—Rhapsody generates all of the source code and configuration files necessary to run a final application on Green Hills Software's royalty-free and POSIX conformant INTEGRITY RTOS. In addition, for resource-constrained and cost-sensitive devices, Green Hills Software's small, fast and royalty-free velOSity microkernel is also supported.

Executive Comments

"By partnering with I-Logix, Green Hills Software is the first to provide a highly integrated requirements-to-deployment solution for embedded systems developers, based on the widely established UML 2.0 and POSIX industry standards. Rhapsody provides the most comprehensive Model-Driven Development solution available in the embedded market today. The combination of Rhapsody with our development tools, royalty-free operating systems and middleware gives our customers an end-to-end solution for optimizing the time-to-market and reliability of their devices."

Dan O'Dowd, Founder and Chief Executive Officer, Green Hills Software

"Green Hills Software offers the industry's most comprehensive IDE and RTOS solutions. When these are combined with our award winning Rhapsody systems design and software development family of products, our two companies, each the fastest growing and most successful in our respective fields, provide a truly synergistic solution. This not only benefits our mutual customers, but also sets a new standard within the embedded systems market by not just integrating products but by also integrating the manner in which these products get to the end-user."

Gene Robinson, President and Chief Executive Officer, I-Logix

Availability

Rhapsody for MULTI is available today from Green Hills Software.

About I-Logix

Founded in 1987, I-Logix is the worldwide leading provider of collaborative Model-Driven Development (MDD) solutions for systems design through software development focused on real-time embedded applications. These solutions allow engineers, operating in either small or very large teams, to graphically model the requirements, behavior, and functionality of embedded systems. The design is iteratively analyzed, validated, and tested throughout the development process while automatically generated production quality code can be output in a variety of languages. I-Logix facilitates team collaboration through unique project and task management capabilities integrated into its UML based MDD solutions, enabling design review and inter-team participation from concept-to-code, regardless of where team members are located. I-Logix is headquartered in Andover, Massachusetts and has sales and support centers throughout North America, Europe and the Far East.

About Green Hills Software

Founded in 1982, Green Hills Software, Inc. is the technology leader in Real-Time Operating Systems (RTOS) and Device Software Optimization (DSO) for 32- and 64-bit embedded systems. Our royalty-free INTEGRITY® RTOS, velOSity™ microkernel, compilers, MULTI® and AdaMULTI™ Integrated Development Environments and TimeMachine™ debugger offer a complete development solution that addresses both deeply embedded and high-reliability applications. Green Hills Software is headquartered in Santa Barbara, CA, with European headquarters in the United Kingdom. Visit Green Hills Software on the web at www.ghs.com.

I-Logix — I-Logix Lands $1,045,000 Aerospace / Defense Order

World's Largest Aerospace / Defense Manufacturer Selects Rhapsody by I-Logix To Push Quality and Productivity Improvements

Nov 18, 2005 - Andover, MA — I-Logix, the leading worldwide Unified Modeling Language (UML™) based Model-Driven Development (MDD) solution provider, announced today that the company landed a $1,045,000 order from one of the world's largest aerospace / defense manufacturers. The aerospace / defense company purchased I-Logix's award winning Rhapsody product family to maximize quality and productivity.

"Every now and then, the planets align and great timing intersects with an extremely competitive product to create a record smashing success like we've had in 2005. And, with this $1M plus order from one of the world's largest military / aerospace companies, the hits just keep on rolling in! Some of our users are focused on time-to-market challenges and Rhapsody clearly saves the day for them. Others, like this company, are more focused on harnessing the energy of thousands of very talented developers with a zero defect quality standard and Rhapsody clearly delivers for them, too. The range of solutions Rhapsody provides is fantastic. This is a great time to be us!" Gene Robinson, CEO I-Logix said.

I-Logix won the contract based on the merits of the company's flagship development environment, Rhapsody. Rhapsody enables engineers to effectively address requirements, design, verification, implementation and test at a higher level of abstraction using a Model-Driven Development (MDD) process versus a textual one. Rhapsody's well recognized key enabling technologies, in addition to its recent expansion to incorporate the Systems Modeling Design Language (SysML) standard and its unique approach to "over-complying" with the Department of Defense Architecture Framework (DoDAF) coupled with the industry's most extensive implementation of UML 2.0 combine to enable teams of all sizes to effectively work together, making for smooth critical design reviews. These benefits, and the ability to generate the full application in C, C++, Ada or Java drastically reduce time to market. Design for testability enables simulation of the design at the earliest stages while providing requirements based testing and the ability to generate tests that provide 100% model coverage, thereby ensuring higher quality designs. The leading aerospace / defense manufacturer recognized that these capabilities all come together in Rhapsody's best in class tool suite, and since their initial roll out, they have realized a 10X improvement in productivity and a major improvement in quality assurance.

Owing to the success of Rhapsody the aerospace / defense manufacturer has extended its use to other major projects including tactical systems, core processing systems, radar, unmanned vehicles and many more at locations throughout the United States.

About I-Logix

Founded in 1987, I-Logix is the worldwide leading provider of Collaborative Model-Driven Development (MDD) solutions for systems design through software development focused on real-time embedded applications. These solutions allow engineers, operating in either small or very large teams, to graphically model the requirements, behavior, and functionality of embedded systems. The design is iteratively analyzed, validated, and tested throughout the development process while automatically generated production quality code can be output in a variety of languages. I-Logix facilitates team collaboration through unique project and task management capabilities integrated into its UML based MDD solutions, enabling design review and inter-team participation from concept-to-code, regardless of where team members are located. I-Logix is headquartered in Andover, Massachusetts and has sales and support centers throughout North America, Europe and the Far East.

Ada and GNU/Linux

Next Debian Ada Compiler

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Date: 5 Oct 2005 08:02:54 -0700
Newsgroups: comp.lang.ada

[See also "About the GNAT GPL 2005 License" in this issue --su]

> In short, if I were Ludovic and were faced with a maintenance task, I'd leave

> 3.15p as it is, as the Ada 95 choice; the GPL version as experimental 0Y with full toolkits, and would aim to replace it in the future with the GMGPL GCC one when it starts to settle and the 0Y feature set is complete.

I'm actually planning to do something close to that. Debian Sarge is the current stable version, and its default Ada compiler is gnat 3.15p. Sarge will remain the current stable version until around December 2006, when Etch is released (the date is tentative: as always, Etch will be released "when it is ready").

I think we will probably end up with GCC 4.1 as the default Ada compiler in Etch. GCC 4.1 may not be perfect WRT Ada 2005, but it will support Ada 95 on more hardware platforms than does GNAT 3.15p; this is a big plus. Also, GCC 4.1 is likely to become the default compiler for C and C++ as well, meaning that it will receive good support on many targets.

I suppose (but this is outside of my control) that GCC 4.1 will first appear in the "experimental" distribution of Debian, which is just designed for such purposes. For those who don't know how Debian works, "experimental" contains a small number of packages that must be installed on top of Sid (unstable). Packages do not migrate from experimental to Sid automatically, but only at their maintainer's explicit request.

Currently, I'm waiting to see how things turn out upstream (i.e. on gcc.gnu.org). I also scan the gcc and gcc-patches lists for Ada- related things. I will announce the beginning of the transition on this forum.

As I have said before, I don't have enough manpower to handle both GNAT GPL and GCC; it is one or the other. The vote that took place earlier means that I will go for GCC and ignore GNAT GPL completely. Unless, of course, there is a landslide of votes in the opposite direction, but this seems unlikely. (I've counted your vote and Marc's).

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 5 Oct 2005 04:10:19 -0700
Subject: Re: GNAT GPL Edition Maintenance and Upgrades
Newsgroups: comp.lang.ada

> I find it quite logical for Debian to just use the standard GCC tree.

Well, latest /= greatest, that's why I kept gnat 3.15p in Debian for so long instead of moving to GCC. Also, GCC is not "standard" by any measure (Ada is not a release criterion for GCC). In contrast, GNAT GPL is "standard" by two measures: it has been blessed by AdaCore, and is known to build ASIS, GLADE, GPS etc. correctly.

> The whole issue is about packaging, not the compiler. I even think that it would be OK to take the GPL edition, replace the offending packages from the version in the GCC tree, and release the whole stuff under GMGPL. But then, someone has to do the packaging…

I contemplated this idea, but when I saw the size of the diff, I backed out. I did "diff -I^-- gcc/gcc/ada gnat-gpl-2005-src/src/ada" (note: ignoring comments and therefore the change of license) and found:

GCC 3.4.4 to GPL: 16.0 megabytes
GCC 4.0.1 to GPL: 11.0 megabytes
GCC HEAD to GPL: 7.1 megabytes

Even with 4.1, the difference is huge. And note that this is only the Ada part of GCC.

[See also "How to Compile GNAT 2005" in this issue --su]

Ada in SuSE Linux

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Fri, 07 Oct 2005 11:11:57 +0200
Subject: SuSE 10.0
Newsgroups: comp.lang.ada

Yesterday I got my new SuSE 10.0. Nice thing about SuSE is that it always comes with an Ada compiler. This time they made a rather big version number jump:

~ Linux root@linux2 Fr Oct 07 10:59:31 standard

> gnat

GNAT 4.0.2 20050901 (prerelease) (SUSE Linux)

Copyright 1996-2005 Free Software Foundation, Inc.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sun, 09 Oct 2005 17:29:09 +0200
Subject: Re: SuSE 10.0
Newsgroups: comp.lang.ada

>> Nice thing about SuSE is that it always comes with an Ada compiler.

> Out of curiosity, what was the version of GCC in the previous version of SuSE?

SuSE 9.x uses GCC 3.3.x.

ASIS and GLADE - Were included in 9.3 but not included in 10.0 any more.

GPS - also no! Which is actually sad.

GDB - yes indeed.

To be honest: I don't think we will get an GMGPL ASIS or GLADE without having access to GNAT/Pro. Recent changes where just to large for the old version to work.

This is why I consider to import the GNAT/GPL sources into the SourceForge projects.

Linuxulator runs GNAT

From: Craig Carey <research@ijs.co.nz>
Date: 2 Sep 2005 09:02:16 -0700
Subject: FreeBSD Linuxulator runs GNAT
Newsgroups: comp.lang.ada

Getting Ada programs compiled inside of the FreeBSD Linuxulator

The FreeBSD Ada Port seems to not run or install or something.

Here is how to compile a Linux gnatmake inside of FreeBSD and have it run under the Linux emulator.

The main trick here is to add this to the compiler options:

```
-largs -static -Wl,
--sysroot=/usr/compat/linux
```

Linux GNAT programs seem to run correctly in the FreeBSD Linuxulator.

(4) In FreeBSD, after the following steps are done:

To get gnatmake to run, this can be added

```
-bargs -E -p -we -s -F

-largs -v -v -static -Wl,
--sysroot=/usr/compat/linux -Wl,
--verbose

gnatmake myprog.adb …... # >| xx
2>&1

brandelf -t Linux myprog # Failed
to eliminate the use of brandelf
```

The -Wl,--verbose seems useful. It lists all /lib files that are used. Use of any FreeBSD *.a file can ruin the executable.

(1) In FreeBSD

** Install a Linux emulator. That involves getting the kernel configured appropriately.

** Install ports/archives/rpm2cpio

** Get the development libc file that matches the other Linuxulator files, to supply libpthread.a.

E.g. download the next from rmpfind.net:

glibc-devel-2.3.2-27.9.7.i386.rpm

** If the Linux rpm was used then the command might look like this:

```
rpm -i --ignoreos --root
/usr/compat/linux --dbpath
/var/lib/rpm $@
```

(That install would copy over too much documentation or fail since Linux kernel headers are missing.)

```
rpm2cpio < glibc-devel-2.3.2-
27.9.7.i386.rpm >| xx

cpio -idm < xx  # create
directories with original time
stamps
```

The "cp -pv" files into /usr/compat/linux/usr/lib/

(2) In Linux, build FSF gcc

Download gcc:
http://gcc.gnu.org/install/download.html

```
export CVS_RSH=ssh
```

```
export
CVSROOT=:ext:anoncvs@savannah.gnu
.org:/cvsroot/gcc

cvs -z9 checkout -P gcc
```

Compile up GCC. It seems that version 4.1.0 has bugs and it is not expected to run correctly, so GCC 3.4.4 can be used.

```
export PATH=/lxg/gcc1/bin:$PATH

cd obj

/fu/@gcc/344/gcc/configure \
 --prefix=/lxg/gcc344    \
 --enable-languages="c,c++,ada" \
 --enable-libada         \
 --with-gcc --with-gnu-ld --with-
gnu-as \
 --enable-threads=posix  \
 --disable-shared   \
 --with-system-zlib \
 --disable-nls      \
 --disable-multilib \
 --disable-libssp

nice make STAGE1_CFLAGS='-O0'
BOOT_CFLAGS='-O0' bootstrap

make -C gcc gnatlib_and_tools  #
try this?

make -C gcc/ada gnatlib        #
try this?

make install
```

(3) In Linux, build Redhat binutils. A version new enough to have the --sysroot feature, is needed.

http://sourceware.org/binutils/

```
export
CVSROOT=:pserver:anoncvs@sourcewa
re.org:/cvs/src
```

Password is "anoncvs", "anonymous"?

```
cvs -z9 login

cvs -z9 co binutils

cd /  ; ln -s / lxsys

# /lxsys is not used again so
presumably "/" could be used
instead

export
PATH=/lxg/gcc344/bin:$PATH_SAVED

export
LD_LIBRARY_PATH=/lxg/gcc344/lib:$
LD_LIBRARY_PATH

# the build modifies the source
directory so it might be copied.

cd obj

lxg/@redhat/src/configure       \
   --prefix=/lxg/gcc344         \
   --with-build-sysroot=/lxsys  \
   --with-sysroot=/lxsys        \
   --enable-static=yes          \
   --enable-languages=c,c++,ada \
   --enable-libada              \
   --with-gcc --with-gnu-ld --
with-gnu-as \
   --enable-threads=posix  \
   --disable-shared     \
   --with-system-zlib   \
   --disable-nls        \
   --disable-multilib   \
   --disable-libssp
```

Then maybe the SuSE GVD or whatever, might be used as the debugger. In Windows, the Gdb debuger of GNAT 3.15p can't handle some Cygwin paths and ver "gdb-5.3" of http://libre.act-europe.fr/GDB/ can fix that (AdaCore names only 6.3, and maybe that is not better)

The above message could have gone to these mailing lists:

GNATLIST: http://hermes.gwu.edu/archives/gnatlist.html

GCC-Help: http://gcc.gnu.org/lists.html

The ARG (run from St Quentin France) is apparently still lacking the thrusting power to get its "arg@ada-auth.org" mailing list to lift off into the night sky of real-time dissemination of information in accordance with deadlines for delays that are under 1.5 minutes…

Here is promotion for running FreeBSD inside of some Windows OS, e.g. from P2P:

NAT connection and the VMware filesharing, FreeBSD guest:

http://www.ijs.co.nz/unix.htm

http://www.ijs.co.nz/code/unix-etc-files.zip

Here are my notes on how to create a defective native FreeBSD cross compiler that targets Linux:

http://www.ijs.co.nz/code/ada95-freebsd-to-linux-cross-compiler.txt

References to Publications

DDC-I Online News

[Extracts from the table of contents. See elsewhere in this news section for selected items. -- su]

From: jc <jcus@ddci.com>
To: 32D September 2005 Online News US <jcus@ddci.com>
Date: Fri, 2 Sep 2005 2:18:01
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News - Real-Time Industry Updates - September 2005, Volume 6, Number 8 - [http://www.ddci.com/news_vol6num8.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

* For Immediate Release

DDC-I Names Bob Morris as President/CEO Joins DDC-I after 5 years as VP Sales & Marketing with LynuxWorks

* For Immediate Release

Upgrade Release for TADS Windows V.6.2.0 Now Available for M68K and 1750A Targets

* Tech Talk

SCORE(R) Gives State-of-the-Art Ada Support to VxWorks

* In The News

Costly Mistakes …Updating the FBI's IT Infrastructure A Look at Three Classic Mistakes from the Agile Software Development Point of View

* Something To Think About: Listen Each Other to a Better Place

The Power of Listening - It Can Improve Life in Many Areas

From: jc <jcus@ddci.com>
To: 33D October 2005 Online News US <jcus@ddci.com>
Date: Mon, 3 Oct 2005 21:45:01
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News

Real-Time Industry Updates - News from DDC-I

October 2005, Volume 6, Number 9 - http://www.ddci.com/news_vol6num9.shtml

A monthly news update dedicated to DDC-I customers & registered subscribers.

* For Immediate Release - State-of-the-Art Ada Support for Wind River Workbench

Seamless Integration - No More Launching One Tool From Another

* Tech Talk

Using Debugger Scripts in SCORE(R) Multi-Language Debugger

* In The News - Ada in Use on Military Aircraft

Better Language for Robust, Long-Standing, Safety-Critical Applications

* Something To Think About: The Frame That Refreshes

It's Not What You Say, It's How You Say It

From: jc <jcus@ddci.com>
To: 34D November 2005 Online News US <jcus@ddci.com>
Date: Wed, 2 Nov 2005 23:18:15
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News

Real-Time Industry Updates - News from DDC-I

November 2005, Volume 6, Number 10 - http://www.ddci.com/news_vol6num10.shtml

A monthly news update dedicated to DDC-I customers & registered subscribers.

* For Immediate Release

Customer Loyalty Brings New Opportunities for DDC-I

* Tech Talk

Optional Arguments of Pragma Import and Export

* In The News

Old Drives Fade Away

* Something To Think About

Humans: The Helpful Species - Working Together as a Team Will Produce Powerful Results

SPARK team newsletter

SPARK news - November 2005

Recent highlights from the SPARK team….

SPARK training

Public course dates for March 2006 are now set. Please see here for details.

We're also in the early stages of planning a public course in Australia - any takers?!?

Ada Europe 2006 Industrial Track Call for Presenters

Ada Europe 2006 will once again include an "industrial track" where you can present your work and projects without the commitment and effort of having to produce a full blown paper.

This is an ideal opportunity for you real users to present some of the amazing work you've been doing with SPARK - I hope that many of you will consider submitting a proposal.

Spreading the word - recent conferences

Rod took part in the NIST SSATTM workshop in Long Beach, California in November. This is a security-related workshop looking at common security vulnerabilities and how static analysis can be used to prevent or detect such things.

On his trip "out west", Rod also visited Microsoft Research, SRI, NASA Ames and the Naval Postgraduate School in Monterey.

Rod attended the "Verified Software: Theories, Tools, and Experiments" workshop in Zurich in October. This was a veritable "who's who" of the program verification community, including none other than Niklaus Wirth as the after-dinner speaker. (If you thinking "who?" then go straight to the back of the class!) Politeness forbids me from passing on what Wirth had to say about the design of C++… :-)

Spreading the word - coming soon

Watch out for SPARK team at the following events:

SPARK team will be presenting a paper about SPARK and SCADE at the Embedded Real Time Software Conference in Toulouse.

SPARK team will be attending the First IEEE International Symposium on Secure Software Engineering (ISSSE) in March 2006 in Washington DC. This looks like being a significant event for high-assurance and security-critical software. We have submitted one technical paper (exciting new results that we're not allowed to talk about just yet!) and several tutorial proposals.

AdaUK Day, March 28th 2006 in Manchester. SPARK Team will be sponsoring, speaking and exhibiting at this event. Our presentation will include highlights of the forthcoming release 7.3 of the SPARK tools, and some material on how we plan to re-introduce generics into the language.

[See also same topic in AUJ 26-2 (Jun 2005), p.82–83 --su]

Ada in the Press

http://www.adacore.com

- Ada Gets First Makeover in a Decade (Software Development Times)

- Ada and the Language Renaissance, by Shannon Cochran. BYTE.com (September 19, 2005)

- Ada used for key systems on military aircraft, by John McHale. Military & Aerospace Electronics (September, 2005)

- Open systems, reliability, and security are primary drivers in software development environments, by John Keller. Military & Aerospace Electronics (November, 2005)

CrossTalk Journal features SPARK again

http://www.praxis-his.com/sparkada

Correctness by Construction in CrossTalk Journal

A new Praxis paper on Correctness by Construction has been published in the December edition of the US CrossTalk Journal.

http://www.stsc.hill.af.mil/crosstalk/2005/12/

[See also "SPARK in IEEE Spectrum Magazine" in AUJ 26-3 (Sep 2005), p.164 --su]

Ada Inside

Swiss — Swiss PostFinance

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Fri, 07 Oct 2005 08:48:28 +0200
Subject: Re: What about big integers in Ada 2005?
Newsgroups: comp.lang.ada

Adrian Hoe wrote:

> I thought Swiss Bank is using some software developed in Ada. If I recall correctly, Paranor was the developer. How did they do that if IS Annex falls short?

Well, it's the Swiss PostFinance. But there is not much interfacing with COBOL - it's all done Ada.

Europe — European Space Agency Ada Usage

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Organization: Tidorum Ltd
Date: Wed, 09 Nov 2005 20:32:39 +0200
Subject: Re: New Ada space project
Newsgroups: comp.lang.ada

> Good to see that the ESA are still using Ada!

[See also "Aonix Tools Selected for European Satellite Launcher" in AUJ 26-2 (Jun 2005) p.84 and "2.2 Billion Miles and Counting: Riding High with Cassini-Huygens" in AUJ 26-1 (Mar 2005) p.63 for to recent examples of Ada usage within ESA --su]

Well, some people within ESA favour Ada, others not. For example, some argue for using Java in the Galileo navigation system.

In my experience (from an earlier life up to about 2003), ESA nowadays accepts the programming language that the prime contractor -- usually one of the larger European space companies such as EADS or Alcatel-Alenia -- chooses for a project. The incidence of C seems to be increasing, for all the usual (poor) reasons.

The last project I worked on had application SW in Ada and low-level I/O drivers in C, "because bit manipulation would be much harder in Ada" according to the subcontractor for the computer HW and driver SW. The next project from the same prime contractor was to be all in C "because this is our strategic decision" according to the prime. I quit. (OK, I had other reasons too.)

> Does anyone have any idea whether it would be easy for a U.S. citizen with five years Ada experience to get a job at the ESA? I'm thinking along the lines of one of the Holland branches (are there others?), but English is my only language.

If you want to write spacecraft software in Ada for ESA projects, you don't want a job with ESA, but with a European space software company, either one of the large "prime" contractors or a smaller subcontractor specialized in software

development. ESA itself does very little SW development, as far as I know. They define and oversee projects but the projects are implemented by contractors.

Companies that do SW work for ESA projects exist in many if not all of the ESA member states; I know of companies in Britain, Ireland, Belgium, Denmark, Norway, the Netherlands, France, Spain, Italy, Germany, Sweden, Finland, Portugal, Austria (listed in no particular order). Pick your place. Most of these companies also do other SW work, few are exclusively space-oriented.

I can't say if U.S. citizenship would usually be an obstacle, but I guess this would depend on whether the company or division also does defence or airplane work, which is more likely for the larger companies. The small company that I worked at (Space Systems Finland Ltd, www.ssf.fi) would not have any problem with U.S. citizens, I believe.

For the language, all ESA project are run in English; all the documentation etc. is in English. Of course, your colleagues may prefer to speak their native language socially. Occasionally you may run into national space projects that merge with or become ESA projects, and then a knowledge of the original national language may be very useful. In one case, I received a SW requirements document that was originally in French, from a French space project, and was now being translated and updated for reusing the SW in an ESA project. At that point in time, the original requirements were in French and the changes were in English, sometimes mixing languages (and acronyms!) in the same sentence, which was a bit confusing (although it meant that change markers were not needed :-) Eventually the doc was translated completely into English, I believe -- I was no longer involved then.

But beware that although English is a certainty, Ada is not. Your first or next ESA project may be Ada, C, Java or who-knows-what.

Indirect Information on Ada Usage

[Extracts from and translations of job-ads and other postings illustrating Ada usage around the world. -- su]

(…) Engineering company expert industrial, scientific and technical software, active in the aeronautic, transportation, automotive, space and avionic domains recruits for new projects.

Joined to your respective sector, your mission will include:

- conception and realization of real-time distributed systems in industrial embedded environments (Ada, C)

- development of software components

- conception and realization of development support tools

- participation in integration and fine-tuning activities.

High education in engineering is required, with a minumum of 1 year of previous experience in a similar function. Knowledge of Ada required and experience with real-time operating systems (VxWorks, QNX, pSOS, RT Linux, …), processors (Power PC, ARM, ST10, …), and language and methods (UML, CVS, RTRT, …). Good knowledge of TCL/TK will constitute a plus.

(…) Engineer to be responsible for full life-cycle development of embedded Command Control Communications real-time application. All development is done in Ada on a Unix platform. BS Computer Science or other related field.

Required Skills

* Must be able to work independently with very minimal supervision.

* Minimum of 5 years of development experience with Ada.

* Minimum of 7 + years of experience in software development.

* Development work with Ada 83 is required.

* Good Object Oriented Skills

* Experience on POSIX.1c or LynxOS or VxWorks or other RTOS preferred.

* Multi-threaded development.

Desired Skills

* Embedded systems development desired.

* Ada 95 is preferred, but not required.

* Experience with communication protocols is a plus.

* Experience with CORBA

* Experience with ClearCase

(…) Responsibilities: Designs, develops, operates, and maintains software components and computing systems software to be applied to and integrated with engineering, scientific, and manufacturing requirements. Applies the appropriate standards, processes, procedures, and tools throughout the system development life cycle to support the generation of such engineering applications and products such as laboratory simulation systems, airplane flight control and display systems, avionics, mechanical and electrical systems, weapons armament, commercial and military aircraft systems, mobile and ground-based defense systems, space systems, and other systems and applications of this nature. Interfaces with customers, suppliers, application users, and other technical and support personnel.

Required skills:

Provide software development of embedded real-time location and electronic warfare systems; Ada83/95; Object-oriented development; VxWorks; Security Clearance; excellent communications, teamwork and interpersonal skills. (…)

Job Responsibilities:

Be responsible for: Design, code and test of real-time embedded software used in various flight critical Engine and Flight control systems.

Required Skills:

Familiar with VAX/VMS and/or Unix based software development tools and debuggers, C/C++, or ADA83/95 and Assembly language skills. Must have real-time embedded experience.

Desired Skills:

* ability to work under tight schedules and meet established deadlines

* excellent oral and written communication skills

* ability to interface effectively with the client and co-workers

* detail oriented, excellent organizational and multi-tasking skills

(…) Job Responsibilities:

(3 positions available; depending upon level of experience with ADA83 and embedded systems. Minimum 3yr – maximum – 7+ yrs) Real time, embedded, avionics Software Engineer needed. Would be responsible for design, development and testing code using Ada software language.

Required Skills:

Ada 83 software development experience and Embedded avionics systems experience

Desired Skills:

* 1553, ARINC 429

* VxWorks

* Link 16

* COM, NAV and/or Controls and Displays

(…) Job Responsibilities:

(2 positions available; Minimum 3yrs of related experience) Position will support development of software for instructional systems, to include graphic user interface (GUI). Determine functional requirements and design, develop, code, integrate and test software to meet those requirements.

Required Skills:

BS in Engineering, Computer Science or equivalent experience in an engineering discipline. Programming knowledge preferred include C, C++, FORTRAN, Ada, Java, and XML with the ability to apply this knowledge in a real-time Unix/Linux environment. Must have good oral and written communication skills.

Ability to work independently or in a team environment.

Desired Skills:

Experience with avionics devices, and/or simulators is preferred.

About Ada jobs

From: Phoebe
Date: Fri, 07 Oct 2005 05:17:35 GMT
Subject: How does one find an Ada job?
Newsgroups: comp.lang.ada

Ada seems like such an elegant, pleasant language to use, but I get the impression that actually getting to use it at work is a distant dream. Should I just not bother actively trying, and relegate it to personal projects?

From: Harald Korneliussen <vintermann@gmail.com>
Date: 19 Oct 2005 01:44:15 -0700
Subject: Re: How does one find an Ada job?
Newsgroups: comp.lang.ada

Robert Klungle wrote:

> All Aerospace companies are looking for Ada software developers desperately Examples are: Raytheon, Boeing, Lockheed Martin, Ball. I know for a fact that Raytheon hires new graduates. Especially if they had a good background in several languages.

I'd like to work with Ada, too, but not for any price. I don't want to work in the arms industry. Are there any other major niches where Ada (or perhaps SPARK or Ravenscar profiles) are used?

I'm taking a course on real-time programming now, and we use the Java RTSJ (well, we would, if we had an implementation. Actually, we just write regular threaded apps :-). I mention it because one of the authors of the RTSJ has written our textbook (Andy Wellings), and he writes in it that RTSJ isn't good enough for safety-critical real-time apps - he recommends the Ravenscar profile as the only option.

So, is it true that for the things Ravenscar are used for, there are no good alternatives?

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 19 Oct 2005 19:55:14 GMT
Subject: Re: How does one find an Ada job?
Newsgroups: comp.lang.ada

Boeing and Airbus both use Ada for their commercial airliners' avionics SW. Raytheon uses Ada for air traffic control SW.

As for arms, they're going to exist whether you work on them or not, and they're going to be controlled by SW. So the important thing, it seems to me, is how reliable that SW is. Do you want to be involved in ensuring the safety of yourself and those you care about, or are you willing to leave that to someone else?

Concurrency in Java is quite low level and error prone, including the RTSJ. Ada is much better, and Ravenscar better still, but for the most critical RT SW, you might want to look at RavenSPARK.

Unfortunately, the lack of good alternatives rarely stops people from using poor alternatives.

From: Anonymous Coward
Date: Wed, 09 Nov 2005 03:33:00 GMT
Subject: Re: How does one find an Ada job?
Newsgroups: comp.lang.ada

The market for Ada is small as it is. Cutting out weapons really shrinks it down. But I would like to think that they are using Ada for safety critical medical hardware (like radiation devices, lasers, etc).

From: Larry Kilgallen <Kilgallen@SpamCop.net>
Newsgroups: comp.lang.ada
Subject: Re: How does one find an Ada job?
Date: 9 Oct 2005 17:22:28 -0500
Organization: LJK Software

Marc A. Criley wrote:

> Identify a market niche.
> Conceive a product that can profitably exploit that niche.
> Develop it in Ada.
> Voila! An Ada job!
> (Then be successful and hire other like-minded individuals.)

But fewer than if you had chosen another language.

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Fri, 07 Oct 2005 10:24:33 +0200
Subject: Re: How does one find an Ada job?
Newsgroups: comp.lang.ada

I get to use it at work. The reason is probably that I am hired to do research, not to develop software. This means that management doesn't interfere with my choices in how I do my work (except when they cost money).

It looks like there are Ada jobs around. But they appear to be in a limited supply - like most interesting jobs.

It probably depends on what you want to do. If you want to do full-time software development in Ada, it will probably be harder, than if you just want a job, where your main task isn't software development, but you are free to write the software you need in Ada.

From: Martin Dowie <martin.dowie@baesystems.com>
Organization: BAE SYSTEMS
Date: Fri, 7 Oct 2005 12:06:40 +0100
Subject: Re: How does one find an Ada job?
Newsgroups: comp.lang.ada

Have you tried:
http://www.adaic.org/jobs/jobs.html?

Ada in Context

Ada 2005 Standarization Status

From: Jean-Pierre Rosen <rosen@adalog.fr>
Newsgroups: fr.comp.lang.ada
Subject: Last news on Ada 2005
Date: Mon, 21 Nov 2005 18:13:59 +0100
Organization: Adalog

At their latest meeting this week-end completed the discussion of all of the last-remaining technical issues. We can thus now regard Ada 2005 as a fully completed language.

A document on it will be circulated real soon. For those who cannot wait, then the current version is at:

http://www.adaic.com/standards/ada05.html

Modulo few corrections to be still made by the ARG authority, that's the future reference manual for Ada 2005.

[See also "Ada 2005 Language Reference Manual" in AUJ 26-3 (Sep 2005), p.166 -- su]

Ada Advantages

From: Jim Rogers <jimmaureenrogers@att.net>
Date: Fri, 07 Oct 2005 04:38:26 GMT
Subject: Re: Investigating Ada
Newsgroups: comp.lang.ada

> I have been looking into Ada as a new language to pick up. I tend to use C for most projects, but I enjoy learning new things. I'm already sold on the value of the language itself from a technical point of view, but my concern is that it might have a rather small userbase, especially without the Ada Mandate. I am having trouble finding recent FAQs or usage information. It's easy to come across something from like 1994, but that's over ten years ago ;) I'm going to learn it anyway, but I'm just hoping that people could comment on its popularity, and also how it has kept up against the other languages which surely must have incorporated a lot of its advantages by now. Thanks

Interestingly, very few languages have incorporated a lot of Ada's advantages by now.

I do not know of any language with as robust and powerful capabilities in concurrency as Ada.

Very few languages allow you to define your own numeric types without also requiring you to explicitly define all the operators for that type.

Very few languages require complete coverage of all values in a case statement.

Very few languages support run-time polymorphism without requiring the use of pointers or references.

Generic programming is common in C++, but introduces a new syntax to the language. Ada's generic capabilities are implemented using standard Ada syntax.

Very few languages provide the automatically defined capabilities of Ada attributes.

Very few languages provide the fine degree of control over data representation provided by Ada.

From: <adaworks@sbcglobal.net>
Date: Sat, 08 Oct 2005 01:49:29 GMT
Subject: Re: Investigating Ada
Newsgroups: comp.lang.ada

Nice list of Ada's benefits in your post. It seems that most programmers are more interested in convenience during development than they are the long-term health of a software product. The latter concern should be a management issue, but not many managers are prepared to understand the problem well enough to choose Ada.

I am currently teaching a graduate seminar titled "Software Evolution." We are examining the issues related to the life cycle of a software system, including how to plan for its continued adaptability to new user requirements, new environments, and new hardware. Sometimes this is thought of as maintenance.

Planning for software evolution, which manifests itself in many forms and technical demands, is becoming more and more essential. One approach is to simply plan to rewrite the code every so often. This is currently a popular choice among those who write software in HTML, XML, scripting languages, etc. However, rewrite is not a good choice for the majority of software systems.

As we examine the language alternatives, Ada comes through as a sound choice. It is becoming increasing clear that, while extensible software can be written in most contemporary languages, the very design of those languages discourages this kind of planned evolution. Java and Eiffel are fairly good. C++, as currently practiced and programmed, is horrible. I suppose C++ is not to blame, but its practitioners seemed determined to prevent anyone from understanding their code well-enough to extend it.

So, as we study the problem of software evolution, Ada is clearly a good choice.

From: Martin Krischik
<krischik@users.sourceforge.net>
Date: Wed, 05 Oct 2005 08:54:39 +0200
Subject: Re: Investigating Ada
Newsgroups: comp.lang.ada

Now that PC become more powerful and therefore can better handle feature rich languages Ada has a little revival.

Mind you: when saying feature rich one take into account the C ISO standard is only a few pages shorter then the Ada ISO standard and C++ has a wooping 200 pages more. The former slim languages have caught up (but they have put on a lot of "backward compatibility fat" instead of "feature muscles").

Commenting Ada Code

From: Niklas Holsti
<niklas.holsti@tidorum.fi>
Newsgroups: comp.lang.ada
Subject: Re: Request for comments on simple Ada program
Date: Wed, 16 Nov 2005 19:08:50 +0200

> The optimal comment/code ratio is zero

On a philosophical level, I agree with you -- writing "comments" is an extra burden, and one risks contradictions between the comments and the code (I recall an article long ago in SIGPLAN Notices with the title "Comments considered harmful" :-)

An ideal language should be expressive and readable enough to stand on its own. Unfortunately, Ada -- or, to be fair, the way I use Ada -- is still too low-level for that. Perhaps some future aspect-oriented language…

> My reason for mentioning what I believe is the optimal comment/code ratio, is actually due to an article in ACM Queue, where the authors used a code quality metric, which equalled more comments with higher code quality.

I agree that such a metric can be very misleading, because then you should measure the quality of the comments, too, which is impossible to automate. But I think anyone who has tried to reuse two code packages, one with (good) comments and the other without, must feel that there is *some* truth in the metric.

> But I don't put the rationale in the source code. The rationale is kept in a separate file. What I still do put in the source code - and don't like putting there - is the abstracts for the subprograms and packages. But how can I easily cross-link the documentation (with the abstracts) and the code (with the implementations)? Should I cite the package name/full subprogram specification in the documentation to create the cross-link? Or can somebody come up with a more elegant solution?

There are some IDE-like commercial tools -- if memory serves, some sort of souped-up requirements databases + design tools + code generatorts with "automatic" document generation functions -- that claim to keep these cross-links for you. But burying my code in that sort of a monster machine makes me very nervous, and they aren't cheap either.

The conversation seems to be drifting towards Knuth's "literate programming". I think it is or was a good idea, but gave more benefit for Pascal than it would give for Ada, which can be more literate in itself. Perhaps the time is ripe for something like "checkable literate programming", modelled on SPARK, where the annotations/comments are readable and informative but also checkable?

From: Samuel Tardieu <sam@rfc1149.net>
Newsgroups: comp.lang.ada
Subject: Re: Request for comments on simple Ada program
Date: 15 Nov 2005 22:53:34 +0100

It depends. Sometimes (particularily when doing embedded stuff), you do have to comment the tricks you resort to. When I do some low-level Ada or Forth programming for microcontrollers, some of the words (that I need to optimize for performance) really need comments. If not for me (the author of the code), at least for others.

What I do in general is reread every code I write one week later. Each time I ask myself "why did I do this?", even for one second, I add a comment. My experience shows that even when I go through the code one year later, I have no question that are not already answered in the comments.

From: Niklas Holsti
<niklas.holsti@tidorum.fi>
Organization: Tidorum Ltd
Newsgroups: comp.lang.ada
Subject: Re: Request for comments on simple Ada program
Date: Wed, 16 Nov 2005 11:03:12 +0200

> Writing the function of the procedure in a comment is OK, although I would like to keep all the non-checkable stuff outside the source file.

People differ. In my Ada code, the "comments" average 33% of non-blank lines (59% for package specs, 23% for package bodies, and note that I never repeat comments from specs in bodies). I still occasionally find it hard to *fully* understand code from a couple of years back. It's not the details -- I know what N := N + 1 means by itself -- but the background assumptions, intentions, limitations, usage rules, interactions.

I put "comments" in quotes above, because I think that this is a bad case of mis-naming in programming language terminology. The word "comment" implies something skimpy, an addition, a note; in my view, what is needed is a rationale, description or motivation that is mainly written *before* the code itself.

As for keeping such text in separate design documents, I would do it if a

customer demanded it, but I think it would be much harder to manage changes, versions and configurations accurately. Still, some very high-level descriptions are nice to keep apart from the source-code, if they are not outdated by day-to-day code changes.

From: Jacob Sparre Andersen <sparre@nbi.dk>
Newsgroups: comp.lang.ada
Subject: Re: Request for comments on simple Ada program
Date: Wed, 16 Nov 2005 15:21:06 +0100

My numbers are 33% in specs and 18% in bodies (22% in total), but I consider them to be too high.

My reason for mentioning what I believe is the optimal comment/code ratio, is actually due to an article in ACM Queue, where the authors used a code quality metric, which equalled more comments with higher code quality.

> I still occasionally find it hard to *fully* understand code from a couple of years back. It's not the details -- I know what N := N + 1 means by itself -- but the background assumptions, intentions, limitations, usage rules, interactions.

That can happen to me too - and to almost any other programmer too, I suppose.

> I put "comments" in quotes above, because I think that this is a bad case of mis-naming in programming language terminology. The word "comment" implies something skimpy, an addition, a note; in my view, what is needed is a rationale, description or motivation that is mainly written *before* the code itself.

But I don't put the rationale in the source code. The rationale is kept in a separate file. What I still do put in the source code - and don't like putting there - is the abstracts for the subprograms and packages. But how can I easily cross-link the documentation (with the abstracts) and the code (with the implementations)? Should I cite the package name/full subprogram specification in the documentation to create the cross-link? Or can somebody come up with a more elegant solution?

> As for keeping such text in separate design documents, I would do it if a customer demanded it, but I think it would be much harder to manage changes, versions and configurations accurately. Still, some very high-level descriptions are nice to keep apart from the source-code, if they are not outdated by day-to-day code changes.

I always keep the design documents separate (when they exist ;-), but I must admit that programming for research purposes is a bit different from commercial software development.

From: Anders Wirzenius <anders@no.email.thanks.invalid>
Subject: Re: Request for comments on simple Ada program
Date: Wed, 16 Nov 2005 09:10:44 GMT
Newsgroups: comp.lang.ada

Samuel Tardieu wrote:

> What I do in general is reread every code I write one week later. Each time I ask myself "why did I do this?", even for one second, I add a comment. My experience shows that even when I go through the code one year later, I have no question that are not already answered in the comments.

Excellent way of working!

Once upon a time (early eighties) when I was a Fortran programmer, my colleague and I used to start coding by writing a pseudo code. The syntax for the pseudo code was Ada. Much of the Ada code was left in the Fortran code as comments.

Conference Calendar

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 *Conference announcements for the international Ada community* at: http://www.cs.kuleuven.ac.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.

2006

January 04-07 — Software Technology Track of the 39th **Hawaii International Conference on System Sciences** (HICSS-39), Kauai, Haway, USA. Includes mini-tracks on: Strategic Software Engineering; Adaptive and Evolvable Software Systems; etc.

January 09-10 — ACM SIGPLAN 2006 **Symposium on Partial Evaluation and Program Manipulation** (PEPM'2006), Charleston, South Carolina, USA

January 11-13 — 33rd Annual ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2006), Charleston, South Carolina, USA. Topics include: fundamental principles and important innovations in the design, definition, analysis, transformation, implementation and verification of programming languages, programming systems, and programming abstractions.

January 14 — 2006 **International Workshop on Foundations and Developments of Object-Oriented Languages** (FOOL/WOOD'2006), Charleston, South Carolina, USA. Following POPL'2006. Topics include: language semantics, type systems, program analysis and verification, concurrent and distributed languages, language-based security, etc.

February 13-17 — 5th **International Conference on COTS-Based Software Systems** (ICCBSS'2006), Orlando, Florida, USA. Theme: "Pushing the COTS Envelope"

♦ February 25-26 — **Free and Open-Source Software Developers' European Meeting** (FOSDEM'2006), Brussels, Belgium. Includes a full-day Ada track on Sunday, February 26, 2006, organized by Ada-Belgium.

March 01-05 — 37th **ACM Technical Symposium on Computer Science Education** (SIGCSE'2006) Houston, TX, USA

March 06-10 — 3rd **International Conference on High Performance Scientific Computing** (HPSC'2006), Hanoi, Vietnam. Topics include: parallel computing (architectures, tools, environments, ...), software development, etc.

☺ March 13-15 — **International Symposium on Secure Software Engineering** (ISSSE'2006), McLean, VA, USA (near Washington, DC). Topics include: Formal specification, designs, policies, and proofs; Coding practices; Static analysis and other automated support; Processes for producing secure software; Certification and accreditation; Relationships among software correctness, reliability, safety, and security; Lessons learned; Technology transfer; etc.

March 20-24 — 5th **International Conference on Aspect-Oriented Software Development** (AOSD'2006), Bonn, Germany. Deadline for submissions: January 23, 2006 (student extravaganza)

March 25-04/02 — **European Joint Conferences on Theory and Practice of Software** (ETAPS'2006), Vienna, Austria

- March 25-26 — 5th **Workshop on Software Composition** (SC'2006)
- Mar 25-Apr 02 — 14th **European Symposium on Programming** (ESOP'2006). Topics include: design of programming languages and calculi and their formal properties; techniques, methods, and tools for their implementation; exploitation of programming styles within different programming paradigms; automatic and manual methods for generating and reasoning about programs; the design and invention of systems and tools to assist in exploitation of the languages

March 27-29 9th **International Conference on Fundamental Approaches to Software Engineering** (FASE'2006). Topics include: Implementation concepts and technologies (distributed and embedded applications), Software evolution (refactoring, reverse and re-engineering, etc.), Software quality (validation and verification of software using theorem proving, testing, analysis, metrics, etc.), Application of formal methods to software development, etc.

April 01-02 4th **Workshop on Quantitative Aspects of Programming Languages** (QAPL'2006). Topics include: design of probabilistic and real-time languages; methodologies for the analysis of probabilistic and timing properties (e.g. security, safety, schedulability); applications to safety-critical systems; etc. Deadline for submissions: February 5, 2006 (extended abstracts)

March 27-30 13th Annual IEEE **International Conference and Workshop on the Engineering of Computer Based Systems** (ECBS'2006), Potsdam, Germany. Theme: Mastering the Complexity of Computer-Based Systems Topics include: Component-Based System Design; Design Evolution; Distributed Systems Design; ECBS Infrastructure (Tools, Environments); Education & Training; Embedded RealTime Software Systems; Formal Methods; Integration Engineering; Modeling and Analysis of Complex Systems; Open Systems; Reliability, Safety, Dependability, Security; Standards; Verification & Validation; etc.

♦ March 28 **Ada Conference 2006 UK**, Manchester, UK

April 02-06 4th **Symposium on Design, Analysis, and Simulation of Distributed Systems** (DASD'2006), Huntsville, Alabama, USA

☺ April 04-07 12th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2006), San Jose, CA, USA. Topics include: programming languages and software engineering for real-time or embedded systems; middleware for real-time or embedded systems; assessments of real-time and embedded technologies for particular application domains; technology transition lessons learned; etc. Deadline for submissions: January 16, 2006 (work in progress papers)

☺ April 10-11 10th **International Conference on Empirical Assessment in Software Engineering** (EASE'2006), Keele University, Staffordshire, UK. Topics include: any aspect of product and process evaluation and assessment, both qualitative and quantitative. Deadline for submissions: January 9, 2006

April 17-19 13th **Annual European Concurrent Engineering Conference** (ECEC'2006), Athens, Greece. Topics include: Engineering of embedded systems, system development process, specification languages; Diagnostics and maintenance, Automated inspection and quality control; Architectures for building CE systems, CE languages and tools, Distributed computing environments; etc. Deadline for submissions: January 20, 2006

☺ April 18-21 1st **EuroSys Conference** (EuroSys'2006), Leuven, Belgium. Topics include: Systems aspects of Programming language support, Distributed algorithms, Middleware, Parallel and concurrent computing, Embedded computers, Real-time computing, Dependable computing, etc. This should be of interest to the European languages community. Deadline for submissions: January 15, 2006 (Roger Needham award)

April 18-21 17th **Australian Software Engineering Conference** (ASWEC'2006), Sydney, Australia. Topics include: Software Design and Patterns; Object-Oriented Software Engineering; Testing, Analysis and Verification; Formal Methods; Software Security, Safety and Reliability; Software Reuse, Components, and Product Line Development; Software Maintenance; Software Engineering Tools; Measurement, Metrics, Experimentation; Technology Transfer, Education; Standards and Legal Issues; etc. Deadline for submissions: February 17, 2006 (industry experience reports)

April 19 19th **Conference on Software Engineering Education and Training** (CSEET'2006), Oahu, Hawaii, USA

☺ April 18 **Workshop on Secure Software Engineering Education & Training** (WSSEET'2006). Topics include: experience, current situation, and future of education and training in software engineering of (more) secure software

April 23-27 21st **ACM Symposium on Applied Computing** (SAC'2006), Dijon, France. Includes tracks on: Software Engineering, etc.

☺ April 23-27 Track on **Programming Languages** (PL'2006). Topics include: Compiling Techniques, Formal Semantics and Syntax, Language Design and Implementation, New Programming Language Ideas and Concepts, Practical Experiences with Programming Languages, Program Analysis and Verification, Program Generation and Transformation, Programming Languages from All Paradigms, etc.

☺ April 23-27 Track on **Object-Oriented Programming Languages and Systems** (OOPS'2006). Topics include: Programming abstractions; Advanced type mechanisms and type safety; Multi-paradigm features; Language features in support of open systems; Program structuring, modularity, generative programming; Distributed Objects and Concurrency; Applications of Distributed Object Computing; etc.

☺ April 24-26 9th IEEE **International Symposium on Object and component-oriented Real-time distributed Computing** (ISORC'2006), Gyeongju, Korea. Topics include: Programming and system engineering (ORC paradigms, languages, RT Corba, UML, application programming interface (API), specification, design, verification, validation, testing, maintenance, system of systems, etc.); System software (real-time kernels, middleware support for ORC, extensibility, scheduling, security, etc.); Applications (embedded systems (automotive, avionics, consumer electronics, etc), real-time object-oriented simulations, etc.); System evaluation (worst-case execution time, dependability, fault detection and recovery time, etc.); ...

April 24-28 30th Annual IEEE/NASA **Software Engineering Workshop** (SEW-30), Columbia, MD, USA. Topics include: Metrics and experience reports; Software quality assurance; Formal methods and formal approaches to software development; Real-time Software Engineering; Software maintenance, reuse, and legacy systems; etc. Deadline for submissions: January 8, 2006

☺ April 25-29 20th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2006), Rhodes Island, Greece. Topics include: all areas of parallel and distributed processing; including the development of experimental or commercial systems; applications of parallel and distributed computing; parallel and distributed software, including parallel programming languages and compilers, runtime systems, middleware, libraries, programming environments and tools, etc.

☺ April 25-26 14th **International Workshop on Parallel and Distributed Real-Time Systems** (WPDRTS'2006). Topics include: Applications, benchmark, and tools; Distributed real-time and embedded middleware; Fault-tolerance and security in real-time systems; Resource management and real-time scheduling; Programming languages and environments; Specification, modeling, and analysis of real-time systems; etc.

May 01-04 **Systems and Software Technology Conference** (SSTC'2006), Salt Lake City, Utah, USA

May 01-05 **International Conference on Practical Software Quality and Testing** (PSQT'2006 West), Las Vegas, NV, USA

May 03-05 6th **International SPICE Conference on Software Process Improvement and Capability dEtermination** (SPICE'2006), Luxembourg, Luxembourg

☺ May 20-28 28th **International Conference on Software Engineering** (ICSE'2006), Shanghai, China

☺ May 22-25 **DAta Systems In Aerospace** (DASIA'2006), Berlin, Germany

May 25-27 **International Conference on Dependability of Computer Systems** (DepCos'2006), Szklarska Poreba, Poland. Topics include: General aspects of dependability; Survivable systems; Coding and dependability; Fault tolerant computing; Software dependability; Software testing, validation and verification; etc.

May 28-31 6th **International Conference on Computational Science** (ICCS'2006), Reading, UK

June 05-09 11th **International Conference on Reliable Software Technologies - Ada-Europe'2006**, Porto, Portugal. Sponsored by Ada-Europe, in cooperation with ACM SIGAda (approval pending). Deadline for submissions: January 12, 2006 (industrial presentations)

☺ June 06-09 **New Technologies for Distributed Systems** (NOTERE'2006), Toulouse, France. Topics include: software components, distributed architectures, models and tools, semi-formal and formal techniques, verification, etc. Deadline for submissions: January 3, 2006 (short and full papers)

June 08-10 2nd **International Conference on Open Source Systems** (OSS'2006), Como, Italy. Topics include: Software engineering perspectives on OSS development, Studies of OSS deployment, etc. Deadline for submissions: January 9, 2006 (research papers, extended abstracts), January 23, 2006 (tutorials, workshops, panels, demonstrations)

☺ June 11-13 5th IFIP **Working Conference on Distributed and Parallel Embedded Systems** (DIPES'2006), Braga, Portugal. Topics include: Design methodology for distributed and parallel embedded systems, Formal verification of embedded systems, Novel programming techniques for distributed real-time systems, Specific (parallel) architectures for distributed embedded systems, Dependability and fault tolerance of distributed embedded systems, Case studies of distributed embedded systems, etc. Deadline for submissions: February 10, 2006

June 12-14 7th **International Conference on Product Focused Software Process Improvement** (PROFES'2006), Amsterdam, The Netherlands. Topics include: Systems and Software Quality, Embedded Systems related Security and Safety, Measurement, SPI in different Software Development Areas, Empirical Studies, Industrial Experiences and Case Studies, Best Practices, Lessons Learned, etc. Deadline for submissions: January 30, 2006 (panels, workshops, tutorials)

June 12-15 9th **International Conference on Software Reuse** (ICSR-9), Torino, Italy. Topics include: Processes to identify and select OTS components; Integration and evolution problems; Reliability and security of OTS components and legal issues; Software generators and domain-specific languages; Evolution of component-based software systems; Benefit and risk analysis of reuse investments; Generation of non-code artefacts; Quality aspects of reuse, e.g. security and reliability; Success and failure stories of reuse approaches from industrial context; etc. Deadline for submissions: January 31, 2006 (full papers)

June 13-16 6th IFIP WG 6.1 **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2006), Bologna, Italy. Co-located with FMOODS'2006 and Coordination'2006. Deadline for submissions: January 10, 2006 (abstracts), January 17, 2006 (full papers), January 31, 2006 (work-in-progress papers), February 28, 2006 (workshop papers)

June 14-16 8th IFIP **International Conference on Formal Methods for Open Object-based Distributed Systems** (FMOODS'2006), Bologna, Italy. Topics include: Semantics and implementation of object-oriented programming and (visual) modelling languages; Formal techniques for specification, design, analysis, verification, validation and testing; Model checking, theorem proving and deductive verification; Model transformations and refactorings; Software architectures; Component-based design; Experience report on best practices and tools; Deadline for submissions: January 10, 2006 (abstracts), January 17, 2006 (papers)

June 19-23 15th IEEE **International Symposium on High-Performance Distributed Computing** (HPDC-15), Paris, France. Topics include: Software environments, programming frameworks & language/compiler support; Fault tolerance, reliability and availability for HPDC applications; etc. Deadline for submissions: January 9, 2006 (abstracts), January 16, 2006 (papers)

June 25-28 2006 **International Conference on Dependable Systems and Networks** (DSN'2006), Philadelphia, PA, USA. Topics include: Dependability Measurement and Analysis; Fault-Tolerance in Distributed and Real-Time Systems; Safety-Critical Systems; Software Reliability; Software Testing, Validation, and Verification; etc. Deadline for submissions: January 14, 2006 (tutorials), April 1, 2006 (student forum, fast abstracts), May 1, 2006 (demonstrations)

☺ June 27 **Workshop on Architecting Dependable Systems** (WADS'2006). Topics include: dependability modeling in software architectures; run-time checks of architectural models; dependability evaluation in software architectures; architectural patterns for dependable systems; exception handling in software architectures; dependable architectures and implementation; etc. Deadline for submissions: March 7, 2006

June 26-28 11th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2006), Bologna, Italy

June 26-29 The 2006 **World Congress in Computer Science, Computer Engineering, and Applied Computing** (WORLDCOMP'2006), Las Vegas, Nevada, USA

☺ June 26-29 **International Conference on Programming Languages and Compilers** (PLC'2006). Topics include: Design and processing of domain specific languages; Implementation of languages features; Language support for security and safety; Compiler construction techniques for modern systems; Program representation & Program analysis; Program optimizations and transformations techniques; Interaction between compilers and architectures; Compilation for distributed, concurrent, and heterogeneous systems; Languages and compilers for high performance computing; Object oriented programming techniques; Object-oriented languages; Run-time environment and storage management techniques; Compilation and interpretation techniques; Code generation and code optimization techniques for modern programming languages; Compilation techniques for embedded code; Security and safety techniques at compiler level; Design of novel language constructs and tool supports; etc. Deadline for submissions: February 20, 2006 (papers)

June 27-30 6th **International Conference on Application of Concurrency to System Design** (ACSD'2006), Turku, Finland. Topics include: design of complex concurrent systems, correct-by-construction design methods and integration of verification techniques with the design process, etc. Deadline for submissions: March 1, 2006 (tool demonstrations)

☺ July 03-07 20th **European Conference on Object-Oriented Programming** (ECOOP'2006), Nantes, France. Topics include: Patterns, Modularity, Adaptability, Separation of Concerns, Components, Frameworks, Concurrency, Real-time, Embedded, Distribution, Domain Specific Languages, Language Workbenches, Multi-paradigm Languages, Language Innovations, Compilation, Methodology, Practices, Metrics, Formal methods, Tools, etc.

☺ July 05-07 18th **Euromicro Conference on Real-Time Systems** (ECRTS'2006), Dresden, Germany. Topics include: all aspects of real-time systems; special focus on industrial applications of real-time technology; compiler support; component-based approaches; middleware and distribution technologies; programming languages; real-time operating systems; model-driven development of embedded RT systems; formal methods; reliability, security and survivability in RT systems; scheduling and schedulability analysis; worst-case execution time analysis; validation techniques; etc.

July 09-16 33rd **International Colloquium on Automata, Languages and Programming** (ICALP'2006), Venice, Italy. Topics include: Principles of Programming Languages, Formal Methods, Models of Concurrent and Distributed Systems, Program Analysis and Transformation, etc. Deadline for submissions: February 10, 2006

☺ July 12-15 12th **International Conference on Parallel and Distributed Systems** (ICPADS'2006), Minneapolis, Minnesota, USA. Topics include: Parallel and Distributed Applications and Algorithms; Reliable and Fault-Tolerant Computing; Real-Time Systems; Tools, and Evaluation; etc. Deadline for submissions: January 15, 2006 (papers)

☺ July 23-26 25th **Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing** (PODC'2006), Denver, Colorado, USA. Topics include: Concurrent programming, Distributed systems and middleware platforms, Correctness and verification of distributed and parallel programming, etc. Deadline for submissions: February 1, 2006

☺ August 14-18 35th **International Conference on Parallel Processing** (ICPP'2006), Columbus, Ohio, USA. Topics include: findings in any aspects of parallel and distributed computing; such as Compilers and Languages, Systems Support for Parallel and Distributed Applications, etc. Deadline for paper submissions: February 1, 2006

August 21-27 14th **International Symposium of Formal Methods Europe** (FM'2006), Hamilton, Canada. Topics include: Tools for formal methods (tool support and software engineering, environments for formal methods), Formal methods in practice (experience with introducing formal methods in industry, case studies), etc. Deadline for submissions: February 24, 2006 (technical papers, workshops, tutorials), May 26, 2006 (posters, tools, doctoral symposium)

☺ August 29-09/01 12th **International Conference on Parallel and Distributed Computing** (Euro-Par'2006), Dresden, Germany. Topics include: the promotion and advancement of parallel computing; Support Tools and

Environments; Distributed Systems and Algorithms; Parallel Programming: Models, Methods, and Languages; Embedded Parallel Systems; etc. Deadline for submissions: January 31, 2006 (full papers)

☺ September 16-20 **Parallel Computing Technologies** (PaCT'2006), Seattle, Washington. USA. Topics include: Compilers and tools for parallel computer systems, Parallel programming languages and applications, Run time system support for parallel systems, Parallel processing in type safe languages, Support for correctnes in hardware and software (esp. with concurrency), etc. Deadline for submissions: March 27, 2006 (abstracts), April 3, 2006 (papers)

October 25-27 5th **International Conference on Software Methodologies Tools, and Techniques** (SoMeT'2006), Quebec, Canada. Topics include: Software methodologies, and tools for robust, reliable, non-fragile software design; Automatic software generation versus reuse, and legacy systems, source code analysis and manipulation; Software evolution techniques; Formal methods for software design; Static and dynamic analysis, and software maintenance; Formal techniques for software representation, software testing and validation; Software reliability, and software diagnosis systems; etc. Deadline for submissions: May 15, 2006

November 12-16 2006 **ACM SIGAda Annual International Conference** (SIGAda'2006), Albuquerque, New Mexico, USA. Sponsored by ACM SIGAda (ACM approval pending). Topics include: reliability needs and styles; safety and high integrity issues; analysis, testing, and validation; standards; use of ASIS for new Ada tool development; mixed-language development; Ada in XML and .NET environments; quality assurance; Ada & software engineering education; commercial Ada applications: what Ada means to the bottom line; static and dynamic code analysis; software architecture and design; etc.

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

2007

June 12th Annual **Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2007), Dundee, Scotland, UK

June 09-16 3rd **History of Programming Languages Conference** (HOPL-III), San Diego, CA, USA. Co-located with FCRC'2007. Deadline for submissions: August 2006 (reworked full papers)

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

2008

June 13th Annual **Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2008), Madrid, Spain

Rationale for Ada 2005: 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 improvements to the predefined library in Ada 2005.

There are a number of important new core packages in Ada 2005. These include a number of packages for the manipulation of various types of containers, packages for directory operations and packages providing access to environment variables.

The entire ISO/IEC 10646:2003 character repertoire is now supported. Program text may now include other alphabets (such as Cyrillic and Greek) and wide-wide characters and strings are supported at run-time. There are also some improvements to the existing character, string and text input–output packages.

The Numerics annex now includes vector and matrix operations including those previously found in the secondary standard ISO/IEC 13813.

Keywords: rationale, Ada 2005.

1 Overview of changes

The WG9 guidance document [1] says

"The main purpose of the Amendment is to address identified problems in Ada that are interfering with Ada's usage or adoption, especially in its major application areas (such as high-reliability, long-lived real-time and/or embedded applications and very large complex systems). The resulting changes may range from relatively minor, to more substantial."

Certainly one of the stated advantages of languages such as Java is that they come with a huge predefined library. By contrast the Ada library is somewhat Spartan and extensions to it should make Ada more accessible.

The guidance document also warns about secondary standards. Its essence is don't use secondary standards if you can get the material into the RM itself. And please put the stuff on vectors and matrices from ISO/IEC 13813 [2] into the RM. The reason for this exhortation is that secondary standards have proved themselves to be almost invisible and hence virtually useless.

We have already discussed the additional library packages in the area of tasking and real time in a previous paper. The following Ada issues cover the relevant changes in other areas and are described in detail in this paper:

161 Preelaborable initialization

248 Directory operations

270 Stream item size control

273 Use of PCS should not be normative

285 Support for 16-bit and 32-bit characters

296 Vector and matrix operations

301 Operations on language-defined strings

302 Container library

328 Non-generic version of Complex_IO

351 Time operations

362 Some predefined packages should be recategorized

366 More liberal rules for Pure units

370 Add standard interface for environment variables

388 Add Greek pi to Ada.Numerics

395 Clarifications concerning 16- and 32-bit characters

400 Wide and wide-wide images

418 Vector norm

427 Default parameters and Calendar operations

428 Input–output for bounded strings

441 Null streams

These changes can be grouped as follows.

First the container library is rather extensive and merits a whole paper alone (302). We only refer to it here for completeness.

New child packages of Calendar provide extra facilities for manipulating times and dates (351, 427).

There are additional packages in the core library providing access to aspects of the operational environment. These concern directory operations (248) and environment variables (370).

There are changes concerning characters both for writing program text itself and for handling characters and strings at run time. There is now support for 16- and 32-bit characters (285, 388, 395, 400), and there are additional operations in the string packages (301, 428).

The Numerics annex is enhanced by the addition of the vector and matrix material previously in ISO/IEC 13813 plus some commonly required linear algebra algorithms (296, 418) and a trivial addition concerning complex input–output (328).

The categorization of various predefined units has been changed in order to remove unnecessary restrictions on their use in distributed systems and similar applications (362, 366). The new pragma Preelaborable_Initialization is introduced as well for similar reasons (161). We can also group a minor change to the Distributed Systems annex here (273).

Finally there is new attribute Stream_Size in order to increase the portability of streams (270) and the parameter Stream of Read, Write etc now has a null exclusion (441).

2 The container library

This is a huge addition to the language and is described in a separate paper for convenience.

3 Times and dates

The first change to note is that the subtype Year_Number in the package Ada.Calendar in Ada 2005 is

```
subtype Year_Number is Integer range 1901 .. 2399;
```

In Ada 95 (and in Ada 83) the range is 1901 .. 2099. This avoids the leap year complexity caused by the 400 year rule at the expense of the use of dates in the far future. But, the end of the 21st century is perhaps not so far into the future, so it was decided that the 2.1k problem should be solved now rather than later. However, it was decided not to change the lower bound because some systems are known to have used that as a time datum. The upper bound was chosen in order to avoid difficulties for implementations. For example, with one nanosecond for Duration'Small, the type Time can just be squeezed into 64 bits.

Having grasped the nettle of doing leap years properly Ada 2005 dives in and deals with leap seconds, time zones and other such matters in pitiless detail.

There are three new child packages Calendar.Time_Zones, Calendar.Arithmetic and Calendar.Formatting. We will look at these in turn.

The specification of the first is

```
package Ada.Calendar.Time_Zones is

   -- Time zone manipulation:
   type Time_Offset is range –28*60 .. 28*60;
   Unknown_Zone_Error: exception;

   function UTC_Time_Offset(Date: Time := Clock)
                                          return Time_Offset;
end Ada.Calendar.Time_Zones;
```

Time zones are described in terms of the number of minutes different from UTC (which curiously is short for Coordinated Universal Time); this is close to but not quite the same as Greenwich Mean Time (GMT) and similarly does not suffer from leaping about in spring and falling about in the autumn. It might have seemed more natural to use hours but some places (for example India) have time zones which are not an integral number of hours different from UTC.

Time is an extraordinarily complex subject. The difference between GMT and UTC is never more than one second but at the moment of writing there is a difference of about 0.577 seconds. The BBC broadcast timesignals based on UTC but call them GMT and with digital broadcasting they turn up late anyway. The chronophile might find the website www.merlyn.demon.co.uk/misctime.htm#GMT of interest.

So the function UTC_Time_Offset applied in an Ada program in Paris to a value of type Time in summer should return a time offset of 120 (one hour for European Central Time plus one hour for daylight saving or heure d'été). Remember that the type Calendar.Time incorporates the date. To find the offset now (that is, at the time of the function call) we simply write

```
Offset := UTC_Time_Offset;
```

and then Clock is called by default.

To find what the offset was on Christmas Day 2000 we write

```
Offset := UTC_Time_Offset(Time_Of(2000, 12, 25));
```

and this should return 60 in Paris. So the poor function has to remember the whole history of local time changes since 1901 and predict them forward to 2399 – these Ada systems are pretty smart! In reality the intent is to use whatever the underlying operating system provides. If the information is not known then it can raise Unknown_Zone_Error.

Note that we are assuming that the package Calendar is set to the local civil (or wall clock) time. It doesn't have to be but one expects that to be the normal situation. Of course it is possible for an Ada system running in California to have Calendar set to the local time in New Zealand but that would be unusual. Equally, Calendar doesn't have to adjust with daylight saving but we expect that it will. (No wonder that Ada.Real_Time was introduced for vital missions such as boiling an egg.)

A useful fact is that

```
Clock – Duration(UTC_Time_Offset*60)
```

gives UTC time – provided we don't do this just as daylight saving comes into effect in which case the call of Clock and that of UTC_Time_Offset might not be compatible.

More generally the type Time_Offset can be used to represent the difference between two time zones. If we want to work with the difference between New York and Paris then we could say

```
NY_Paris: Time_Offset := –360;
```

The time offset between two different places can be greater than 24 hours for two reasons. One is that the International Date Line weaves about somewhat and the other is that daylight saving time can extend the difference as well. Differences of 26 hours can easily occur and 27 hours is possible. Accordingly the range of the type Time_Offset allows for a generous 28 hours.

The package Calendar.Arithmetic provides some awkward arithmetic operations and also covers leap seconds. Its specification is

```
package Ada.Calendar.Arithmetic is

   -- Arithmetic on days:
   type Day_Count is range
     –366*(1+Year_Number'Last – Year_Number'First)
     ..
     +366*(1+Year_Number'Last – Year_Number'First);

   subtype Leap_Seconds_Count is
                              Integer range –2047 .. 2047;

   procedure Difference(Left, Right: in Time;
              Days: out Day_Count; Seconds: out Duration;
                 Leap_Seconds: out Leap_Seconds_Count);

   function "+" (Left: Time; Right: Day_Count)
                                                 return Time;
   function "+" (Left: Day_Count; Right: Time)
                                                 return Time;
   function "–" (Left: Time; Right: Day_Count)
                                                 return Time;
   function "–" (Left, Right: Time) return Day_Count;

end Ada.Calendar.Arithmetic;
```

The range for Leap_Seconds_Count is generous. It allows for a leap second at least four time a year for the foreseeable future – the somewhat arbitrary range chosen allows the value to be accommodated in 12 bits. And the 366 in Day_Count is also a bit generous – but the true expression would be very unpleasant.

One of the problems with the old planet is that it is slowing down and a day as measured by the Earth's rotation is now a bit longer than 86,400 seconds. Naturally enough we have to keep the seconds uniform and so in order to keep worldly clocks synchronized with the natural day, an odd leap second has to be added from time to time. This is always added at midnight UTC (which means it can pop up in the middle of the day in other time zones). The existence of leap seconds makes calculations with times somewhat tricky.

The basic trouble is that we want to have our cake and eat it. We want to have the invariant that a day has 86,400 seconds but unfortunately this is not always the case.

The procedure Difference operates on two values of type Time and gives the result in three parts, the number of days (an integer), the number of seconds as a Duration and the number of leap seconds (an integer). If Left is later then Right then all three numbers will be nonnegative; if earlier, then nonpositive.

Remember that Difference like all these other operations always works on local time as defined by the clock in Calendar (unless stated otherwise).

Suppose we wanted to find the difference between noon on June 1st 1982 and 2pm on July 1st 1985 according to a system set to UTC. We might write

```
Days: Day_Count;
Secs: Duration;
Leaps: Leap_Seconds_Count;
...
Difference(Time_Of(1985, 7, 1, 14*3600.0),
                Time_Of(1982, 6, 1, 12*3600.0),
                        Days, Secs, Leaps);
```

The results should be

```
Days = 365+366+365+30 = 1126,
Secs = 7200.0,
Leaps = 2.
```

There were leap seconds on 30 June 1983 and 30 June 1985.

The functions "+" and "–" apply to values of type Time and Day_Count (whereas those in the parent Calendar apply only to Time and Duration and thus only work for intervals of a day or so). Note that the function "–" between two values of type Time in this child package produces the same value for the number of days as the corresponding call of the function Difference – leap seconds are completely ignored. Leap seconds are in fact ignored in all the operations "+" and "–" in the child package.

However, it should be noted that Calendar."–" counts the true seconds and so the expression

```
Calendar."–" (Time_Of(1985, 7, 1, 1*3600.0),
                Time_Of(1985, 6, 30, 23*3600.0))
```

has the Duration value 7201.0 and not 7200.0 because of the leap second at midnight that night. (We are assuming that our Ada system is running at UTC.) The same calculation in New York will produce 7200.0 because the leap second doesn't occur until 4 am in EST (with daylight saving).

Note also that

```
Calendar."–" (Time_Of(1985, 7, 1, 0.0),
                Time_Of(1985, 6, 30, 0.0))
```

in Paris where the leap second occurs at 10pm returns 86401.0 whereas the same calculation in New York will return 86400.0.

The third child package Calendar.Formatting has a variety of functions. Its specification is

```
with Ada.Calendar.Time_Zones;
use Ada.Calendar.Time_Zones;
package Ada.Calendar.Formatting is

   -- Day of the week:
   type Day_Name is (Monday, Tuesday, Wednesday,
                     Thursday, Friday, Saturday, Sunday);

   function Day_Of_Week(Date: Time) return Day_Name;

   -- Hours:Minutes:Seconds access:
   subtype Hour_Number is Natural range 0 .. 23;
   subtype Minute_Number is Natural range 0 .. 59;
   subtype Second_Number is Natural range 0 .. 59;
```

```
   subtype Second_Duration is Day_Duration
                                    range 0.0 .. 1.0;

   function Year(Date: Time;
                 Time_Zone: Time_Offset := 0)
                                    return Year_Number;

   -- similarly functions Month, Day, Hour, Minute

   function Second(Date: Time) return Second_Number;

   function Sub_Second(Date: Time)
                                    return Second_Duration;

   function Seconds_Of(Hour: Hour_Number;
                       Minute: Minute_Number;
                       Second: Second_Number := 0;
                       Sub_Second: Second_Duration := 0.0)
                                    return Day_Duration;

   procedure Split(Seconds: in Day_Duration;       -- (1)
                   Hour: out Hour_Number;
                   Minute: out Minute_Number;
                   Second: out Second_Number;
                   Sub_Second: out Second_Duration);

   procedure Split(Date: in Time;                  -- (2)
                   Year: out Year_Number;
                   Month: out Month_Number;
                   Day: out Day_Number;
                   Hour: out Hour_Number;
                   Minute: out Minute_Number;
                   Second: out Second_Number;
                   Sub_Second: out Second_Duration;
                   Time_Zone: in Time_Offset := 0);

   function Time_Of(Year: Year_Number;
                    Month: Month_Number;
                    Day: Day_Number;
                    Hour: Hour_Number;
                    Minute: Minute_Number;
                    Second: Second_Number;
                    Sub_Second: Second_Duration := 0.0;
                    Leap_Second: Boolean := False;
                    Time_Zone: Time_Offset := 0)
                                    return Time;

   function Time_Of(Year: Year_Number;
                    Month: Month_Number;
                    Day: Day_Number;
                    Seconds: Day_Duration;
                    Leap_Second: Boolean := False;
                    Time_Zone: Time_Offset := 0)
                                    return Time;

   procedure Split(Date: in Time;                  -- (3)
                   ... -- as (2) but with additional parameter
                   Leap_Second: out Boolean;
                   Time_Zone: in Time_Offset := 0);

   procedure Split(Date: in Time;                  -- (4)
                   ... -- as Calendar.Split
                   ... -- but with additional parameter
                   Leap_Second: out Boolean;
                   Time_Zone: in Time_Offset := 0);

   -- Simple image and value:
   function Image(Date: Time;
                  Include_Time_Fraction: Boolean := False;
                  Time_Zone: Time_Offset := 0)
                                    return String;

   function Value(Date: String;
                  Time_Zone: Time_Offset := 0)
                                    return Time;

   function Image (Elapsed_Time: Duration;
                   Include_Time_Fraction: Boolean := False)
                                    return String;

   function Value(Elapsed_Time: String) return Duration;

end Ada.Calendar.Formatting;
```

The function Day_Of_Week will be much appreciated. It is a nasty calculation.

Then there are functions Year, Month, Day, Hour, Minute, Second and Sub_Second which return the corresponding parts of a Time taking account of the time zone given as necessary. It is unfortunate that functions returning the parts of a time (as opposed to the parts of a date) were not provided in Calendar originally. All that Calendar provides is Seconds which gives the number of seconds from midnight and leaves users to chop it up for themselves. Note that Calendar.Second returns a Duration whereas the function in this child package is Seconds which returns an Integer. The fraction of a second is returned by Sub_Second.

Most of these functions have an optional parameter which is a time zone offset. Wherever in the world we are running, if we want to know the hour according to UTC then we write

```
Hour(Clock, UTC_Time_Offset)
```

If we are in New York and want to know the hour in Paris then we write

```
Hour(Clock, –360)
```

since New York is 6 hours (360 minutes) behind Paris.

Note that Second and Sub_Second do not have the optional Time_Offset parameter because offsets are an integral number of minutes and so the number of seconds does not depend upon the time zone.

The package also generously provides four procedures Split and two procedures Time_Of. These have the same general purpose as those in Calendar. There is also a function Seconds_Of. We will consider them in the order of declaration in the package specification above.

The function Seconds_Of creates a value of type Duration from components Hour, Minute, Second and Sub_Second. Note that we can use this together with Calendar.Time_Of to create a value of type Time. For example

```
T := Time_Of(2005, 4, 2, Seconds_Of(22, 4, 10, 0.5));
```

makes the time of the instant when I typed that last semicolon.

The first procedure Split is the reverse of Seconds_Of. It decomposes a value of type Duration into Hour, Minute, Second and Sub_Second. It is useful with the function Calendar.Split thus

```
Split(Some_Time, Y, M, D, Secs);          -- split time
Split(Secs, H, M, S, SS);                 -- split secs
```

The next procedure Split (no 2) takes a Time and a Time_Offset (optional) and decomposes the time into its seven components. Note that the optional parameter is last for convenience. The normal rule for parameters of predefined procedures is that parameters of mode in are first and parameters of mode out are last. But this is a nuisance if parameters of mode in have defaults since this forces named notation if the default is used.

There are then two functions Time_Of which compose a Time from its various constituents and the Time_Offset (optional). One takes seven components (with individual Hour, Minute etc) whereas the other takes just four components (with Seconds in the whole day). An interesting feature of these two functions is that they also have a Boolean parameter Leap_Second which by default is False.

The purpose of this parameter needs to be understood carefully. Making up a typical time will have this parameter as False. But suppose we need to compose the time midway through the leap second that occurred on 30 June 1985 and assign it to a variable Magic_Moment. We will assume that our Calendar is in New York and set to EST with daylight saving (and so midnight UTC is 8pm in New York). We would write

```
Magic_Moment: Time :=
      Time_Of(1985, 6, 30, 19, 59, 59, 0.5, True);
```

In a sense there were two 19:59:59 that day in New York. The proper one and then the leap one; the parameter distinguishes them. So the moment one second earlier is given by

```
Normal_Moment: Time :=
      Time_Of(1985, 6, 30, 19, 59, 59, 0.5, False);
```

We could have followed ISO and used 23:59:60 UTC and so have subtype Second_Number **is** Natural **range** 0 .. 60; but this would have produced an incompatibility with Ada 95.

Note that if the parameter Leap_Second is True and the other parameters do not identify a time of a leap second then Time_Error is raised.

There are then two corresponding procedures Split (nos 3 and 4) with an out parameter Leap_Second. One produces seven components and the other just four. The difference between this seven-component procedure Split (no 3) and the earlier Split (no 2) is that this one has the out parameter Leap_Second whereas the other does not. Writing

```
Split(Magic_Moment, 0, Y, M, D, H, M, S, SS, Leap);
```

results in Leap being True whereas

```
Split(Normal_Moment, 0, Y, M, D, H, M, S, SS, Leap);
```

results in Leap being False but gives all the other out parameters (Y, ... , SS) exactly the same values.

On the other hand calling the version of Split (no 2) without the parameter Leap_Second thus

```
Split(Magic_Moment, 0, Y, M, D, H, M, S, SS);
Split(Normal_Moment, 0, Y, M, D, H, M, S, SS);
```

produces exactly the same results.

The reader might wonder why there are two Splits on Time with Leap_Second but only one without. This is because the parent package Calendar already has the other one (although without the time zone parameter). Another point is that in the case of Time_Of, we can default the Leap parameter being of mode in but in the case of Split the parameter has mode out and cannot be omitted. It would be bad practice to encourage the use of a dummy parameter which is ignored and hence there have to be additional versions of Split.

Finally, there are two pairs of functions Image and Value. The first pair works with values of type Time. A call of Image returns a date and time value in the standard ISO 8601 format. Thus taking the Normal_Moment above

```
Image(Normal_Moment)
```

returns the following string

```
"1985-06-30 19:59:59"          -- in New York
```

If we set the optional parameter Include_Time_Fraction to True thus

```
Image(Normal_Moment, True)
```

then we get

```
"1985-06-30 19:59:59.50"
```

There is also the usual optional Time_Zone parameter so we could produce the time in Paris (from the program in New York) thus

```
Image(Normal_Moment, True, –360)
```

giving

```
"1985-07-01 02:59:59.50"          -- in Paris
```

The matching Value function works in reverse as expected.

We would expect to get exactly the same results with Magic_Moment. However, since some implementations might have an ISO function available in their operating system it is also allowed to produce

```
"1985-06-30 19:59:60"          -- in New York
```

The other Image and Value pair work on values of type Duration thus

```
Image(10_000.0)                -- "02:46:40"
```

with the optional parameter Include_Time_Fraction as before. Again the corresponding function Value works in reverse.

4 Operational environment

Two new packages are added to Ada 2005 in order to aid communication with the operational environment. They are Ada.Environment_Variables and Ada.Directories.

The package Ada.Environment_Variables has the following specification

```
package Ada.Environment_Variables is
   pragma Preelaborate(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, Value: in String));

end Ada.Environment_Variables;
```

This package provides access to environment variables by name. What this means and whether it is supported depends upon the implementation. But most operating systems have environment variables of some sort. And if not, the implementation is encouraged to simulate them.

The values of the variable are also implementation defined and so simply represented by strings.

The behaviour is straightforward. We might check to see if there is a variable with the name "Ada" and then read and print her value and set it to 2005 if it is not, thus

```
if not Exists("Ada") then
   raise Horror;                    -- quel dommage!
end if;

Put("Current value of Ada is ");  Put_Line(Value("Ada"));

if Value("Ada") /= "2005" then
   Put_Line("Revitalizing Ada now");
   Set("Ada", "2005");
end if;
```

The procedure Clear with a parameter deletes the variable concerned. Thus Clear("Ada") eliminates her completely so that a subsequent call Exists("Ada") will return False. Note that Set actually clears the variable concerned and then defines a new one with the given name and value. The procedure Clear without a parameter clears all variables.

We can iterate over the variables using the procedure Iterate. For example we can print out the current state by

```
procedure Print_One(Name, Value: in String) is
begin
   Put_Line(Name & "=" & Value);
end Print_One;
...
Iterate(Print_One'Access);
```

The procedure Print_One prints the name and value of the variable passed as parameters. We then pass an access to this procedure as a parameter to the procedure Iterate and Iterate then calls Print_One for each variable in turn.

Note that the slave procedure has both Name and Value as parameters. It might be thought that this was unnecessary since the user can always call the function Value. However, real operating systems can sometimes have several variables with the same name; providing two parameters ensures that the name/value pairs are correctly matched.

Attempting to misuse the environment package such as reading a variable that doesn't exist raises Constraint_Error or Program_Error.

There are big dangers of race conditions because the environment variables are really globally shared. Moreover, they might be shared with the operating system itself as well as programs written in other languages.

A particular point is that we must not call the procedures Set or Clear within a procedure passed as a parameter to Iterate.

The other environment package is Ada.Directories. Its specification is

```
with Ada.IO_Exceptions;
with Ada.Calendar;
package Ada.Directories is

   -- Directory and file operations:
   function Current_Directory return String;
   procedure Set_Directory(Directory: in String);
   procedure Create_Directory(New_Directory: in String;
                              Form: in String := "");
   procedure Delete_Directory(Directory: in String);
   procedure Create_Path(New_Directory: in String;
                         Form: in String := "");
   procedure Delete_Tree(Directory: in String);
   procedure Delete_File(Name: in String);
   procedure Rename(Old_Name: in String;
                    New_Name: in String);
   procedure Copy_File(Source_Name: in String;
         Target_Name: in String; Form: in String := "");

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

   -- File and directory queries:
   type File_Kind is
                (Directory, Ordinary_File, Special_File);
   type File_Size is range 0 .. implementation_defined;
   function Exists(Name: String) return Boolean;
   function Kind(Name: String) return File_Kind;
   function Size(Name: String) return File_Size;
   function Modification_Time(Name: String)
                         return Ada.Calendar.Time;
```

```
   -- Directory searching:
   type Directory_Entry_Type is limited private;
   type Filter_Type is array (File_Kind) of Boolean;
   type Search_Type is limited private;
   procedure Start_Search(Search: in out Search_Type;
                          Directory: in String; Pattern: in String;
                          Filter: in Filter_Type := (others => True));
   procedure End_Search(Search: in out Search_Type);
   function More_Entries(Search: Search_Type)
                                                 return Boolean;
   procedure Get_Next_Entry
                                (Search: in out Search_Type;
                      Directory_Entry: out Directory_Entry_Type);
   procedure Search(Directory: in String;
                    Pattern: in String;
                    Filter: in Filter_Type := (others => True);
                    Process: not null access procedure
                   (Directory_Entry: in Directory_Entry_Type));

   -- Operations on Directory Entries:
   function Simple_Name(Directory_Entry:
                        Directory_Entry_Type) return String;
   function Full_Name(Directory_Entry:
                        Directory_Entry_Type) return String;
   function Kind(Directory_Entry: Directory_Entry_Type)
                                                 return File_Kind;
   function Size(Directory_Entry: Directory_Entry_Type)
                                                 return File_Size;
   function Modification_Time(Directory_Entry:
         Directory_Entry_Type) return Ada.Calendar.Time;

   Status_Error: exception renames
                              Ada.IO_Exceptions.Status_Error;
   Name_Error: exception renames
                              Ada.IO_Exceptions.Name_Error;
   Use_Error: exception renames
                               Ada.IO_Exceptions.Use_Error;
   Device_Error: exception renames
                             Ada.IO_Exceptions.Device_Error;
private
   -- Not specified by the language
end Ada.Directories;
```

Most operating systems have some sort of tree-structured filing system. The general idea of this package is that it allows the manipulation of file and directory names as far as is possible in a unified manner which is not too dependent on the implementation and operating system.

Files are classified as directories, special files and ordinary files. Special files are things like devices on Windows and soft links on Unix; these cannot be created or read by the predefined Ada input–output packages.

Files and directories are identified by strings in the usual way. The interpretation is implementation defined.

The full name of a file is a string such as

```
"c:\adastuff\rat\library.doc"
```

and the simple name is

```
"library.doc"
```

At least that is in good old DOS. In Windows XP it is something like

```
"C:\Documents and Settings\john.JBI3\
                My Documents\adastuff\rat\library.doc"
```

For the sake of illustration we will continue with the simple DOS example. The current directory is that set by the "cd" command. So assuming we have done

```
c:\>cd adastuff
c:\adastuff>
```

then the function Current_Directory will return the string "c:\adastuff". The procedure Set_Directory sets the current default directory. The procedures Create_Directory and Delete_Directory create and delete a single directory. We can either give the full name or just the part starting from the current default. Thus

```
Create_Directory("c:\adastuff\history");
Delete_Directory("history");
```

will cancel out.

The procedure Create_Path creates several nested directories as necessary. Thus starting from the situation above, if we write

```
Create_Path("c:\adastuff\books\old");
```

then it will first create a directory "books" in "c:\adastuff" and then a directory "old" in "books". On the other hand if we wrote Create_Path("c:\adastuff\rat"); then it would do nothing since the path already exists. The procedure Delete_Tree deletes a whole tree including subdirectories and files.

The procedures Delete_File, Rename and Copy_File behave as expected. Note in particular that Copy_File can be used to copy any file that could be copied using a normal input–output package such as Text_IO. For example, it is really tedious to use Text_IO to copy a file intact including all line and page terminators. It is a trivial matter using Copy_File.

Note also that the procedures Create_Directory, Create_Path and Copy_File have an optional Form parameter. Like similar parameters in the predefined input–output packages the meaning is implementation defined.

The next group of six functions, Full_Name, Simple_Name, Containing_Directory, Extension, Base_Name and Compose just manipulate strings representing file names and do not in any way interact with the actual external file system. Moreover, of these, only the behaviour of Full_Name depends upon the current directory.

The function Full_Name returns the full name of a file. Thus assuming the current directory is still "c:\adastuff"

```
Full_Name("rat\library.doc")
```

returns "c:\adastuff\rat\library.doc" and

```
Full_Name("library.doc")
```

returns "c:\adastuff\library.doc". The fact that such a file does not exist is irrelevant. We might be making up the

name so that we can then create the file. If the string were malformed in some way (such as "66##77") so that the corresponding full name if returned would be nonsense then Name_Error is raised. But Name_Error is never raised just because the file does not exist.

On the other hand

```
Simple_Name("c:\adastuff\rat\library.doc")
```

returns "library.doc" and not "rat\library.doc". We can also apply Simple_Name to a string that does not go back to the root. Thus

```
Simple_Name("rat\library.doc");
```

is allowed and also returns "library.doc".

The function Containing_Directory_Name removes the simple name part of the parameter. We can even write

```
Containing_Directory_Name("..\rat\library.doc")
```

and this returns "..\rat"; note that it also removes the separator "\".

The functions Extension and Base_Name return the corresponding parts of a file name thus

```
Base_Name("rat\library.doc")          -- "library"
Extension("rat\library.doc")          -- "doc"
```

Note that they can be applied to a simple name or to a full name or, as here, to something midway between.

The function Compose can be used to put the various bits together, thus

```
Compose("rat", "library", "doc")
```

returns "rat\library.doc". The default parameters enable bits to be omitted. In fact if the third parameter is omitted then the second parameter is treated as a simple name rather than a base name. So we could equally write

```
Compose("rat","library.doc")
```

The next group of functions, Exists, Kind, Size and Modification_Time act on a file name (that is the name of a real external file) and return the obvious result. (The size is measured in stream elements – usually bytes.)

Various types and subprograms are provided to support searching over a directory structure for entities with appropriate properties. This can be done in two ways, either as a loop under the direct control of the programmer (sometimes called an active iterator) or via an access to subprogram parameter (often called a passive iterator). We will look at the active iterator approach first.

The procedures Start_Search, End_Search and Get_Next_Entry and the function More_Entries control the search loop. The general pattern is

```
Start_Search( ... );
while More_Entries( ... ) loop
  Get_Next_Entry( ... );
  ...              -- do something with the entry found
end loop;
End_Search( ... );
```

Three types are involved. The type Directory_Entry_Type is limited private and acts as a sort of handle to the entries found. Valid values of this type can only be created by a call of Get_Next_Entry whose second parameter is an out parameter of the type Directory_Entry_Type. The type Search_Type is also limited private and contains the state of the search. The type Filter_Type provides a simple means of identifying the kinds of file to be found. It has three components corresponding to the three values of the enumeration type File_Kind and is used by the procedure Start_Search.

Suppose we want to look for all ordinary files with extension "doc" in the directory "c:\adastuff\rat". We could write

```
Rat_Search: Search_Type;
Item: Directory_Entry_Type;
Filter: Filter_Type := (Ordinary_File => True,
                                        others => False);
...
Start_Search(Rat_Search, "c:\adastuff\rat", "*.doc", Filter);
while More_Entries(Rat_Search) loop
  Get_Next_Entry(Rat_Search, Item);
  ...              -- do something with Item
end loop;
End_Search(Rat_Search);
```

The third parameter of Start_Search (which is "*.doc" in the above example) represents a pattern for matching names and thus provides further filtering of the search. The interpretation is implementation defined except that a null string means match everything. However, we would expect that writing "*.doc" would mean search only for files with the extension "doc".

The alternative mechanism using a passive iterator is as follows. We first declare a subprogram such as

```
procedure Do_It(Item: in Directory_Entry_Type) is
begin
  ...              -- do something with item
end Do_It;
```

and then declare a filter and call the procedure Search thus

```
Filter: Filter_Type := (Ordinary_File => True,
                                        others => False);
...
Search("c:\adastuff\rat", "*.doc", Filter, Do_It'Access);
```

The parameters of Search are the same as those of Start_Search except that the first parameter of type Search_Type is omitted and a final parameter which identifies the procedure Do_It is added. The variable Item which we declared in the active iterator is now the parameter Item of the procedure Do_It.

Each approach has its advantages. The passive iterator has the merit that we cannot make mistakes such as forget to call End_Search. But some find the active iterator easier to understand and it can be easier to use for parallel searches.

The final group of functions enables us to do useful things with the results of our search. Thus Simple_Name and Full_Name convert a value of Directory_Entry_Type to the corresponding simple or full file name. Having obtained the file name we can do everything we want but for convenience the functions Kind, Size and Modification_Time are provided which also directly take a parameter of Directory_Entry_Type.

So to complete this example we might print out a table of the files found giving their simple name, size and modification time. Using the active approach the loop might then become

```
while More_Entries(Rat_Search) loop
   Get_Next_Entry(Rat_Search, Item);
   Put(Simple_Name(Item));  Set_Col(15);
   Put(Size(Item/1000));  Put(" KB");  Set_Col(25);
   Put_Line(Image(Modification_Time(Item)));
end loop;
```

This might produce a table such as

```
access.doc      152 KB    2005-04-05  09:03:10
containers.doc  372 KB    2005-06-14  21:39:05
general.doc     181 KB    2005-03-03  08:43:15
intro.doc       173 KB    2004-11-25  15:52:20
library.doc     149 KB    2005-04-08  13:50:05
oop.doc         179 KB    2005-02-25  18:34:55
structure.doc   151 KB    2005-04-05  09:09:25
tasking.doc     174 KB    2005-03-31  11:16:40
```

Note that the function Image is from the package Ada.Calendar.Formatting discussed in the previous section.

Observe that the search is carried out on the directory given and does not look at subdirectories. If we want to do that then we can use the function Kind to identify subdirectories and then search recursively.

It has to be emphasized that the package Ada.Directories is very implementation dependent and indeed might not be supported by some implementations at all. Implementations are advised to provide any additional useful functions concerning retrieving other information about files (such as name of the owner or the original creation date) in a child package Ada.Directories.Information.

Finally, note that misuse of the various operations will raise one of the exceptions Status_Error, Name_Error, Use_Error or Device_Error from the package IO_Exceptions.

5 Characters and strings

An important improvement in Ada 2005 is the ability to deal with 16-and 32-bit characters both in the program text and in the executing program.

The fine detail of the changes to the program text are perhaps for the language lawyer. The purpose is to permit the use of all relevant characters of the entire ISO/IEC 10646:2003 repertoire. The most important effect is that we can write programs using Cyrillic, Greek and other character sets.

A good example is provided by the addition of the constant

```
π: constant := Pi;
```

to the package Ada.Numerics. This enables us to write mathematical programs in a more natural notation thus

```
Circumference: Float := 2.0 * π * Radius;
```

Other examples might be for describing polar coordinates thus

```
R: Float := Sqrt(X*X + Y*Y);
θ: Angle := Arctan(Y, X);
```

and of course in France we can now declare a decent set of ingredients for breakfast

```
type Breakfast_Stuff is (Croissant, Café, Œuf, Beurre);
```

Curiously, although the ligature æ is in Latin-1 and thus available in Ada 95 in identifiers, the ligature œ is not (for reasons we need not go into). However, in Ada 95, œ is a character of the type Wide_Character and so even in Ada 95 one can order breakfast thus

```
Put("Deux œufs easy-over avec jambon"); -- wide string
```

In order to manipulate 32-bit characters, Ada 2005 includes types Wide_Wide_Character and Wide_Wide_String in the package Standard and the appropriate operations to manipulate them in packages such as

```
Ada.Strings.Wide_Wide_Bounded
Ada.Strings.Wide_Wide_Fixed
Ada.Strings.Wide_Wide_Maps
Ada.Strings.Wide_Wide_Maps.Wide_Wide_Constants
Ada.Strings.Wide_Wide_Unbounded
Ada.Wide_Wide_Text_IO
Ada.Wide_Wide_Text_IO.Text_Streams
Ada.Wide_Wide_Text_IO.Complex_IO
Ada.Wide_Wide_Text_IO.Editing
```

There are also new attributes Wide_Wide_Image, Wide_Wide_Value and Wide_Wide_Width and so on.

The addition of wide-wide characters and strings introduces many additional possibilities for conversions. Just adding these directly to the existing package Ada.Characters.Handling could cause ambiguities in existing programs when using literals. So a new package Ada.Characters.Conversions has been added. This contains conversions in all combinations between Character, Wide_Character and Wide_Wide_Character and similarly for strings. The existing functions from Is_Character to To_Wide_String in Ada.Characters.Handling have been banished to Annex J.

The introduction of more complex writing systems makes the definition of the case insensitivity of identifiers, (the equivalence between upper and lower case), much more complicated.

In some systems, such as the ideographic system used by Chinese, Japanese and Korean, there is only one case, so things are easy. But in other systems, like the Latin, Greek and Cyrillic alphabets, upper and lower case characters have to be considered. Their equivalence is usually

straightforward but there are some interesting exceptions such as

- Greek has two forms for lower case sigma (the normal form σ and the final form ς which is used at the end of a word). These both convert to the one upper case letter Σ.
- German has the lower case letter ß whose upper case form is made of two letters, namely SS.
- Slovenian has a grapheme LJ which is considered a single letter and has three forms: LJ, Lj and lj.

The Greek situation used to apply in English where the long s was used in the middle of words (where it looked like an f but without a cross stroke) and the familiar short s only at the end. To modern eyes this makes poetic lines such as "Where the bee sucks, there suck I" somewhat dubious. (This is sung by Ariel in Act V Scene I of The Tempest by William Shakespeare.)

The definition chosen for Ada 2005 closely follows those provided by ISO/IEC 10646:2003 and by the Unicode Consortium; this hopefully means that all users should find that the case insensitivity of identifiers works as expected in their own language.

Of interest to all users whatever their language is the addition of a few more subprograms in the string handling packages. As explained in the Introduction, Ada 95 requires rather too many conversions between bounded and unbounded strings and the raw type String and, moreover, multiple searching is inconvenient.

The additional subprograms in the packages are as follows.

In the package Ada.Strings.Fixed (assuming **use** Maps; for brevity)

```
function Index(Source: String; Pattern: String;
      From: Positive; Going: Direction := Forward;
      Mapping: Character_Mapping := Identity)
                                        return Natural;

function Index(Source: String; Pattern: String;
      From: Positive; Going: Direction := Forward;
      Mapping: Character_Mapping_Function)
                                        return Natural;

function Index(Source: String; Set: Character_Set;
      From: Positive; Test: Membership := Inside;
      Going: Direction := Forward) return Natural;

function Index_Non_Blank(Source: String;
      From: Positive; Going: Direction := Forward)
                                        return Natural;
```

The difference between these and the existing functions is that these have an additional parameter From. This makes it much easier to search for all the occurrences of some pattern in a string.

Similar functions are also added to the packages Ada.Strings.Bounded and Ada.Strings.Unbounded.

Thus suppose we want to find all the occurrences of "bar" in the string "barbara barnes" held in the variable BS of type Bounded_String. (I have put my wife into lower case for convenience.) There are 3 of course. The existing function Count can be used to determine this fact quite easily

```
N := Count(BS, "bar")              -- is 3
```

But we really need to know where they are; we want the corresponding index values. The first is easy in Ada 95

```
I := Index(BS, "bar")              -- is 1
```

But to find the next one in Ada 95 we have to do something such as take a slice by removing the first three characters and then search again. This would destroy the original string so we need to make a copy of at least part of it thus

```
Part := Delete(BS, I, I+2);        -- 2 is length "bar" – 1
I := Index(Part, "bar") + 3;       -- is 4
```

and so on in the not-so-obvious loop. (There are other ways such as making a complete copy first, this could either be in another bounded string or perhaps it is simplest just to copy it into a normal String first; but whatever we do it is messy.) In Ada 2005, having found the index of the first in I, we can find the second by writing

```
I := Index(BS, "bar", From => I+3);
```

and so on. This is clearly much easier.

The following are also added to Ada.Strings.Bounded

```
procedure Set_Bounded_String
      (Target: out Bounded_String;
      Source: in String; Drop: in Truncation := Error);

function Bounded_Slice(Source: Bounded_String;
                Low: Positive; High: Natural)
                                  return Bounded_String;

procedure Bounded_Slice(Source: in Bounded_String;
                Target: out Bounded_String;
                Low: in Positive; High: in Natural);
```

The procedure Set_Bounded_String is similar to the existing function To_Bounded_String. Thus rather than

```
BS := To_Bounded_String("A Bounded String");
```

we can equally write

```
Set_Bounded_String(BS, "A Bounded String");
```

The slice subprograms avoid conversion to and from the type String. Thus to extract the characters from 3 to 9 we can write

```
BS := Bounded_Slice(BS, 3, 9);          -- "Bounded"
```

whereas in Ada 95 we have to write something like

```
BS := To_Bounded(Slice(BS, 3, 9));
```

Similar subprograms are added to Ada.Strings.Unbounded. These are even more valuable because unbounded strings are typically implemented with controlled types and the use of a procedure such as Set_Unbounded_String is much

more efficient than the function To_Unbounded_String because it avoids assignment and thus calls of Adjust.

Input and output of bounded and unbounded strings in Ada 95 can only be done by converting to or from the type String. This is both slow and untidy. This problem is particularly acute with unbounded strings and so Ada 2005 provides the following additional package (we have added a use clause for brevity as usual)

```
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
package Ada.Text_IO.Unbounded_IO is

   procedure Put(File: in File_Type;
                 Item: in Unbounded_String);
   procedure Put(Item: in Unbounded_String);

   procedure Put_Line(File: in File_Type;
                      Item: in Unbounded_String);
   procedure Put_Line(Item: in Unbounded_String);

   function Get_Line(File: File_Type)
                     return Unbounded_String;
   function Get_Line return Unbounded_String;

   procedure Get_Line(File: in File_Type;
                      Item: out Unbounded_String);
   procedure Get_Line(Item: out Unbounded_String);

end Ada.Text_IO.Unbounded_IO;
```

The behaviour is as expected.

There is a similar package for bounded strings but it is generic. It has to be generic because the package Generic_Bounded_Length within Strings.Bounded is itself generic and has to be instantiated with the maximum string size. So the specification is

```
with Ada.Strings.Bounded; use Ada.Strings.Bounded;
generic
   with package Bounded is new
                     Generic_Bounded_Length(<>);
   use Bounded;
package Ada.Text_IO.Bounded_IO is

   procedure Put(File: in File_Type;
                 Item: in Bounded_String);
   procedure Put(Item: in Bounded_String);

   ... -- etc as for Unbounded_IO

end Ada.Text_IO.Bounded_IO;
```

It will be noticed that these packages include functions Get_Line as well as procedures Put_Line and Get_Line corresponding to those in Text_IO. The reason is that procedures Get_Line are not entirely satisfactory.

If we do successive calls of the procedure Text_IO.Get_Line using a string of length 80 on a series of lines of length 80 (we are reading a nice old deck of punched cards), then it does not work as expected. Alternate calls return a line of characters and a null string (the history of this behaviour goes back to early Ada 83 days and is best left dormant).

Ada 2005 accordingly adds corresponding functions Get_Line to the package Ada.Text_IO itself thus

```
function Get_Line(File: File_Type) return String;
function Get_Line return String;
```

Successive calls of a function Get_Line then neatly return the text on the cards one by one without bother.

6 Numerics annex

When Ada 95 was being designed, the Numerics Rapporteur Group pontificated at length over what features should be included in Ada 95 itself, what should be placed in secondary standards, and what should be left to the creativeness of the user community.

A number of secondary standards had been developed for Ada 83. They were

11430 Generic package of elementary functions for Ada,

11729 Generic package of primitive functions for Ada,

13813 Generic package of real and complex type declarations and basic operations for Ada (including vector and matrix types),

13814 Generic package of complex elementary functions for Ada.

The first two, 11430 and 11729, were incorporated into the Ada 95 core language. The elementary functions, 11430, (Sqrt, Sin, Cos etc) became the package Ada.Numerics.Generic_Elementary_Functions in A.5.1, and the primitive functions, 11729, became the various attributes such as Floor, Ceiling, Exponent and Fraction in A.5.3. The original standards were withdrawn long ago.

The other two standards, although originally developed as Ada 83 standards did not become finally approved until 1998.

In the case of 13814, the functionality was all incorporated into the Numerics annex of Ada 95 as the package Ada.Numerics.Generic_Complex_Elementary_Functions in G.1.2. Accordingly the original standard has now lapsed.

However, the situation regarding 13813 was not so clear. It covered four areas

1 a complex types package including various complex arithmetic operations,

2 a real arrays package covering both vectors and matrices,

3 a complex arrays package covering both vectors and matrices, and

4 a complex input–output package.

The first of these was incorporated into the Numerics annex of Ada 95 as the package Ada.Numerics.Generic_Complex_Types in G.1.1 and the last similarly became the package Ada.Text_IO.Complex_IO in G.1.3. However, the array packages, both real and complex, were not incorporated into Ada 95.

The reason for this omission is explained in Section G.1.1 of the rationale for Ada 95 [3] which says

> A decision was made to abbreviate the Ada 95 packages by omitting the vector and matrix types and operations. One reason was that such types and operations were largely self-evident, so that little real help would be provided by defining them in the language. Another reason was that a future version of Ada might add enhancements for array manipulation and so it would be inappropriate to lock in such operations permanently.

The sort of enhancements that perhaps were being anticipated were facilities for manipulating arbitrary subpartitions of arrays such as were provided in Algol 68. These rather specialized facilities have not been added to Ada 2005 and indeed it seems most unlikely that they would ever be added. The second justification for omitting the vector and matrix facilities of 13813 thus disappears.

In order to overcome the objection that everything is self-evident we have taken the approach that we should further add some basic facilities that are commonly required, not completely trivial to implement, but on the other hand are mathematically well understood.

So the outcome is that Ada 2005 includes almost everything from 13813 plus subprograms for

- finding the norm of a vector,
- solving sets of linear equations,
- finding the inverse and determinant of a matrix,
- finding the eigenvalues and eigenvectors of a symmetric real or Hermitian matrix.

A small number of operations that were not related to linear algebra were removed (such as raising all elements of a matrix to a given power).

So Ada 2005 includes two new packages which are Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays. It would take too much space to give the specifications of both in full so we give just an abbreviated form of the real package in which the specifications of the usual operators are omitted thus

```
generic
   type Real is digits <>;
package Ada.Numerics.Generic_Real_Arrays is
   pragma Pure(Generic_Real_Arrays);

   -- Types
   type Real_Vector is array (Integer range <>)
                                          of Real'Base;
   type Real_Matrix is array (Integer range <>,
                              Integer range <>) of Real'Base;

   -- Real_Vector arithmetic operations
   ... -- unary and binary "+" and "–" giving a vector
   ... -- also inner product and two versions of "abs" – one
   ... -- returns a vector and the other a value of Real'Base

   -- Real_Vector scaling operations
   ... -- operations "*" and "/" to multiply a vector by a
   ... -- scalar and divide a vector by a scalar

   -- Other Real_Vector operations
   function Unit_Vector(Index: Integer; Order: Positive;
                        First: Integer := 1) return Real_Vector;

   -- Real_Matrix arithmetic operations
   ... -- unary "+", "–", "abs", binary "+", "–" giving a matrix
   ... -- "*" on two matrices giving a matrix, on a vector
   ... -- and a matrix giving a vector, outer product of two
   ... -- vectors giving a matrix, and of course
   function Transpose(X: Real_Matrix)
                                          return Real_Matrix;

   -- Real_Matrix scaling operations
   ... -- operations "*" and "/" to multiply a matrix by a
   ... -- scalar and divide a matrix by a scalar

   -- Real_Matrix inversion and related operations
   function Solve(A: Real_Matrix; X: Real_Vector)
                                          return Real_Vector;
   function Solve(A, X: Real_Matrix) return Real_Matrix;
   function Inverse(A: Real_Matrix) return Real_Matrix;
   function Determinant(A: Real_Matrix)
                                          return Real'Base;

   -- Eigenvalues and vectors of a real symmetric matrix
   function Eigenvalues(A: Real_Matrix)
                                          return Real_Vector;
   procedure Eigensystem(A: in Real_Matrix;
      Values: out Real_Vector; Vectors: out Real_Matrix);

   -- Other Real_Matrix operations
   function Unit_Matrix(Order: Positive;
         First_1, First_2: Integer := 1) return Real_Matrix;

end Ada.Numerics.Generic_Real_Arrays;
```

Many of these operations are quite self-evident. The general idea as far as the usual arithmetic operations are concerned is that we just write an expression in the normal way as illustrated in the Introduction. But the following points should be noted.

There are two operations "**abs**" applying to a Real_Vector thus

```
function "abs"(Right: Real_Vector) return Real_Vector;
function "abs"(Right: Real_Vector) return Real'Base;
```

One returns a vector each of whose elements is the absolute value of the corresponding element of the parameter (rather boring) and the other returns a scalar which is the so-called L2-norm of the vector. This is the square root of the inner product of the vector with itself or $\sqrt{(\Sigma x_i x_i)}$ – or just $\sqrt{(x_i x_i)}$ using the summation convention (which will be familiar to those who dabble in the relative world of tensors). This is provided as a distinct operation in order to avoid any intermediate overflow that might occur if the user were to compute it directly using the inner product "*".

There are two functions Solve for solving one and several sets of linear equations respectively. Thus if we have the single set of *n* equations

$$\boldsymbol{Ax} = \boldsymbol{y}$$

then we might write

```
X, Y: Real_Vector(1 .. N);
A: Real_Matrix(1 .. N, 1 .. N);
...
Y := Solve(A, X);
```

and if we have *m* sets of *n* equations we might write

```
XX, YY: Real_Matrix(1 .. N, 1 .. M)
A: Real_Matrix(1 .. N, 1 .. N);
...
YY := Solve(A, XX);
```

The functions Inverse and Determinant are provided for completeness although they should be used with care. Remember that it is foolish to solve a set of equations by writing

```
Y := Inverse(A)*X;
```

because it is both slow and prone to errors. The main problem with Determinant is that it is liable to overflow or underflow even for moderate sized matrices. Thus if the elements are of the order of a thousand and the matrix has order 10, then the magnitude of the determinant will be of the order of 10^{30}. The user may therefore have to scale the data.

Two subprograms are provided for determining the eigenvalues and eigenvectors of a symmetric matrix. These are commonly required in many calculations in domains such as elasticity, moments of inertia, confidence regions and so on. The function Eigenvalues returns the eigenvalues (which will be non-negative) as a vector with them in decreasing order. The procedure Eigensystem computes both eigenvalues and vectors; the parameter Values is the same as that obtained by calling the function Eigenvalues and the parameter Vectors is a matrix whose columns are the corresponding eigenvectors in the same order. The eigenvectors are mutually orthonormal (that is, of unit length and mutually orthogonal) even when there are repeated eigenvalues. These subprograms apply only to symmetric matrices and if the matrix is not symmetric then Argument_Error is raised.

Other errors such as the mismatch of array bounds raise Constraint_Error by analogy with built-in array operations.

The reader will observe that the facilities provided here are rather humble and presented in a simple black-box style. It is important to appreciate that we do not see the Ada predefined numerics library as being in any way in competition with or as a substitute for professional libraries such as the renowned BLAS (Basic Linear Algebra Subprograms, see www.netlib.org/blas). Indeed our overall goal is twofold

- to provide commonly required simple facilities for the user who is not a numerical professional,
- to provide a baseline of types and operations that forms a firm foundation for binding to more general facilities such as the BLAS.

We do not expect users to apply the operations in our Ada packages to the huge matrices that arise in areas such as partial differential equations. Such matrices are often of a special nature such as banded and need the facilities of a comprehensive numerical library. We have instead striven to provide easy to use facilities for the programmer who has a small number of equations to solve such as might arise in navigational applications.

Simplicity is evident in that functions such as Solve do not reveal the almost inevitable underlying LU decomposition or provide parameters controlling for example whether additional iterations should be applied. However, implementations are advised to apply an additional iteration and should document whether they do or not.

Considerations of simplicity also led to the decision not to provide automatic scaling for the determinant or to provide functions for just the largest eigenvalue and so on.

Similarly we only provide for the eigensystems of symmetric real matrices. These are the ones that commonly arise and are well behaved. General nonsymmetric matrices can be troublesome.

Appropriate accuracy requirements are specified for the inner product and L2-norm operations. Accuracy requirements for Solve, Inverse, Determinant, Eigenvalues and Eigenvectors are implementation defined which means that the implementation must document them.

The complex package is very similar and will not be described in detail. However, the generic formal parameters are interesting. They are

```
with Ada.Numerics.Generic_Real_Arrays,
                Ada.Numerics.Generic_Complex_Types;
generic
  with package Real_Arrays is
          new Ada.Numerics.Generic_Real_Arrays(<>);
  use Real_Arrays;
  with package Complex_Types is
     new Ada.Numerics.Generic_Complex_Types(Real);
  use Complex_Types;
package Ada.Numerics.Generic_Complex_Arrays is
  ...
```

Thus we see that it has two formal packages which are the corresponding real array package and the existing Ada 95 complex types and operations package. The formal parameter of the first is <> and that of the second is Real which is exported from the first package and ensures that both are instantiated with the same floating point type.

As well as the obvious array and matrix operations, the complex package also has operations for composing complex arrays from cartesian and polar real arrays, and computing the conjugate array by analogy with scalar operations in the complex types package. There are also mixed real and complex array operations but not mixed imaginary, real and complex array operations. Altogether the complex array package declares some 80 subprograms (there are around 30 in the real array package) and adding

imaginary array operations would have made the package unwieldy (and the reference manual too heavy).

By analogy with real symmetric matrices, the complex package has subprograms for determining the eigensystems of Hermitian matrices. A Hermitian matrix is one whose complex conjugate equals its transpose; such matrices have real eigenvalues and are well behaved.

We conclude this discussion of the Numerics annex by mentioning one minute change regarding complex input–output. Ada 2005 includes preinstantiated forms of Ada.Text_IO.Complex_IO such as Ada.Complex_Text_IO (for when the underlying real type is the type Float), Ada.Long_Complex_Text_IO (for type Long_Float) and so on. These are by analogy with Float_Text_IO, Long_Float_Text_IO and their omission from Ada 95 was probably an oversight.

7 Categorization of library units

It will be recalled that library units in Ada 95 are categorized into a hierarchy by a number of pragmas thus

```
pragma Pure( ... );
pragma Shared_Passive( ... );
pragma Remote_Types( ... );
pragma Remote_Call_Interface( ... );
```

Each category imposes restrictions on what the unit can contain. An important rule is that a unit can only depend on units in the same or higher categories (the bodies of the last two are not restricted).

The pragmas Shared_Passive, Remote_Types, and Remote_Call_Interface concern distributed systems and thus are rather specialized. A minor change made in the 2001 Corrigendum was that the pragma Remote_Types was added to the package Ada.Finalization in order to support the interchange of controlled types between partitions in a distributed system.

Note that the pragma Preelaborate does not fit into this hierarchy. In fact there is another hierarchy thus

```
pragma Pure( ... );
pragma Preelaborate( ... );
```

and again we have the same rule that a unit can only depend upon units in the same or higher category. Thus a pure unit can only depend upon other pure units and a preelaborable unit can only depend upon other preelaborable or pure units.

A consequence of this dual hierarchy is that a shared passive unit cannot depend upon a preelaborable unit – the units upon which it depends have to be pure or shared passive and so on for the others. However, there is a separate rule that a unit which is shared passive, remote types or RCI must itself be preelaborable and so has to also have the pragma Preelaborate.

The categorization of individual predefined units is intended to make them as useful as possible. The stricter the category the more useful the unit because it can be used in more circumstances.

The categorization was unnecessarily weak in Ada 95 in some cases and some changes are made in Ada 2005.

The following packages which had no categorization in Ada 95 have pragma Preelaborate in Ada 2005

```
Ada.Asynchronous_Task_Control
Ada.Dynamic_Priorities
Ada.Exceptions
Ada.Synchronous_Task_Control
Ada.Tags
Ada.Task_Identification
```

The following which had pragma Preelaborate in Ada 1995 have been promoted to pragma Pure in Ada 2005

```
Ada.Characters.Handling
Ada.Strings.Maps
Ada.Strings.Maps.Constants
System
System.Storage_Elements
```

These changes mean that certain facilities such as the ability to analyse exceptions are now available to preelaborable units. Note however, that Wide_Maps and Wide_Maps.Wide_Constants stay as preelaborable because they may be implemented using access types.

Just for the record the following packages (and functions, Hash is a function) which are new to Ada 2005 have the pragma Pure

```
Ada.Assertions
Ada.Characters.Conversions
Ada.Containers
Ada.Containers.Generic_Array_Sort
Ada.Containers.Generic_Constrained_Array_Sort
Ada.Dispatching
Ada.Numerics.Generic_Real_Arrays
Ada.Numerics.Generic_Complex_Arrays
Ada.Strings.Hash
```

And the following new packages and functions have the pragma Preelaborate

```
Ada.Containers.Doubly_Linked_Lists
Ada.Containers.Hashed_Maps
Ada.Containers.Hashed_Sets
Ada.Containers.Ordered_Maps
Ada.Containers.Ordered_Sets
Ada.Containers.Vectors
Ada.Environment_Variables
Ada.Strings.Unbounded_Hash
Ada.Strings.Wide_Wide_Maps
Ada.Strings.Wide_Wide_Maps.Wide_Wide_Constants
Ada.Tags.Generic_Dispatching_Constructor
Ada.Task_Termination
```

plus the indefinite containers as well.

A problem with preelaborable units in Ada 95 is that there are restrictions on declaring default initialized objects in a unit with the pragma Preelaborate. For example, we cannot

declare objects of a private type at the library level in such a unit. This is foolish for consider

```
package P is
  pragma Preelaborate(P);
  X: Integer := 7;
  B: Boolean := True;
end;
```

Clearly these declarations can be preelaborated and so the package P can have the pragma Preelaborate. However, now consider

```
package Q is
  pragma Preelaborate(Q);        -- legal
  type T is private;
private
  type T is
    record
      X: Integer := 7;
      B: Boolean := True;
    end record;
end Q;

with Q;
package P is
  pragma Preelaborate(P);        -- illegal
  Obj: Q.T;
end P;
```

The package Q is preelaborable because it does not declare any objects. However, the package P is not preelaborable because it declares an object of the private type T – the theory being of course that since the type is private we do not know that its default initial value is static.

This is overcome in Ada 2005 by the introduction of the pragma Preelaborable_Initialization. Its syntax is

```
pragma Preelaborable_Initialization(direct_name);
```

We can now write

```
package Q is
  pragma Preelaborate(Q);
  type T is private;
  pragma Preelaborable_Initialization(T);
private
  type T is
    record
      X: Integer := 7;
      B: Boolean := True;
    end record;
end Q;
```

The pragma promises that the full type will have preelaborable initialization and the declaration of the package P above is now legal.

The following predefined private types which existed in Ada 95 have the pragma Preelaborable_Initialization in Ada 2005

```
Ada.Exceptions.Exception_Id
Ada.Exceptions.Exception_Occurrence
Ada.Finalization.Controlled
Ada.Finalization.Limited_Controlled
Ada.Numerics.Generic_Complex_Types.Imaginary
Ada.Streams.Root_Stream_Type
Ada.Strings.Maps.Character_Mapping
Ada.Strings.Maps.Character_Set
Ada.Strings.Unbounded.Unbounded_String
Ada.Tags.Tag
Ada.Task_Identification.Task_Id
Interfaces.C.Strings.chars_ptr
System.Address
System.Storage_Pool.Root_Storage_Pool
```

Wide and wide-wide versions also have the pragma as appropriate. Note that it was not possible to apply the pragma to Ada.Strings.Bounded.Generic_Bounded_Length. Bounded_String because it would have made it impossible to instantiate Generic_Bounded_Length with a non-static expression for the parameter Max.

The following private types which are new in Ada 2005 also have the pragma Preeleborable_Initialization

```
Ada.Containers.Vectors.Vector
Ada.Containers.Vectors.Cursor
Ada.Containers.Doubly_Linked_Lists.List
Ada.Containers.Doubly_Linked_Lists.Cursor
Ada.Containers.Hashed_Maps.Map
Ada.Containers.Hashed_Maps.Cursor
Ada.Containers.Ordered_Maps.Map
Ada.Containers.Ordered_Maps.Cursor
Ada.Containers.Hashed_Sets.Set
Ada.Containers.Hashed_Sets.Cursor
Ada.Containers.Ordered_Sets.Set
Ada.Containers.Ordered_Sets.Cursor
```

and similarly for the indefinite containers.

A related change concerns the definition of pure units. In Ada 2005, pure units can now use access to subprogram and access to object types provided that no storage pool is created.

Finally, we mention a small but important change regarding the partition communication subsystem System.RPC. Implementations conforming to the Distributed Systems annex are not required to support this predefined interface if another interface would be more appropriate – to interact with CORBA for example.

8 Streams

Important improvements to the control of streams were described in the paper on the object oriented model where we discussed the new package Ada.Tags.Generic_ Dispatching_Constructor and various changes to the parent package Ada.Tags itself. In this section we mention two other changes.

There is a problem with the existing stream attributes such as (see RM 13.13.2)

```
procedure S'Write
  (Stream: access Root_Stream_Type'Class; Item: in T);
```

where S is a subtype of T. Note that for the parameter Item, its type *T* is in italic and so has to be interpreted according to the kind of type. In the case of integer and enumeration types it means that the parameter Item has type T'Base.

Given a declaration such as

```
type Index is range 1 .. 10;
```

different implementations might use different representations for Index'Base – some might use 8 bits others might use 32 bits and so on.

Now stream elements themselves are typically 8 bits and so with an 8-bit base, there will be one value of Index per stream element whereas with a 32-bit base each value of Index will take 4 stream elements. Clearly a stream written by the 8-bit implementation cannot be read by the 32-bit one.

This problem is overcome in Ada 2005 by the introduction of a new attribute Stream_Size. The universal integer value S'Stream_Size gives the number of bits used in the stream for values of the subtype S. We are guaranteed that it is a multiple of Stream_Element'Size. So the number of stream elements required will be

```
S'Stream_Size / Stream_Element'Size
```

We can set the attribute in the usual way provided that the value given is a static multiple of Stream_Element'Size. So in the case above we can write

```
for Index'Stream_Size use 8;
```

and portability is then assured. That is provided that Stream_Element_Size is 8 anyway and that the implementation accepts the attribute definition clause (which it should).

A minor change is that the parameter Stream of the various atttributes now has a null exclusion so that S'Write is in fact

```
procedure S'Write
  (Stream: not null access Root_Stream_Type'Class;
   Item: in T);
```

Perhaps surprisingly this does not introduce any incompatibilities since in Ada 95 passing null raises Constraint_Error anyway and so this change just clarifies the situation.

On this dullish but important topic here endeth the Rationale for Ada 2005 apart from various exciting appendices and an extensive subpaper on containers.

References

[1] ISO/IEC JTC1/SC22/WG9 N412 (2002) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment*.

[2] ISO/IEC 13813:1997 (1997) Generic packages of real and complex type declarations and basic operations for Ada (including vector and matrix types).

[3] *Ada 95 Rationale* (1995) LNCS 1247, Springer-Verlag.

[4] J. G. P. Barnes (1998) *Programming in Ada 95*, 2nd ed., Addison-Wesley.

Rationale for Ada 2005: 6a Containers

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 the predefined container library in Ada 2005.

Keywords: rationale, Ada 2005.

1 Organization of containers

A major enhancement to the predefined library in Ada 2005 is the addition of a container library. This is quite extensive and merits this separate paper on its own. Other aspects of the predefined library and the overall rationale for extending the library were described in the previous paper.

The main packages in the container library can be grouped in various ways. One set of packages concerns the manipulation of objects of definite types and another, essentially identical, set concerns indefinite types. (Remember that an indefinite (sub)type is one for which we cannot declare an object without giving a constraint.) The reason for the duplication concerns efficiency. It is much easier to manipulate definite types and although the packages for indefinite types can be used for definite types, this would be rather inefficient.

We will generally only consider the definite packages. These in turn comprise two groups.

Sequence containers – these hold sequences of elements. There are packages for manipulating vectors and for manipulating linked lists. These two packages have much in common. But they have different behaviours in terms of efficiency according to the pattern of use. In general (with some planning) it should be possible to change from one to the other with little effort.

Associative containers – these associate a key with each element and then store the elements in order of the keys. There are packages for manipulating hashed maps, ordered maps, hashed sets and ordered sets. These four packages also have much in common and changing between hashed and ordered versions is usually feasible.

There are also quite separate generic procedures for sorting arrays which we will consider later.

The root package is

```
package Ada.Containers is
   pragma Pure(Containers);

   type Hash_Type is mod implementation-defined;
   type Count_Type is range 0 .. implementation-defined;

end Ada.Containers;
```

The type Hash_Type is used by the associative containers and Count_Type is used by both kinds of containers typically for the number of elements in a container. Note that we talk about elements in a container rather than the components in a container – components is the Ada term for the items of an array or record as an Ada type and it is convenient to use a different term since in the case of containers the actual data structure is hidden.

Worst-case and average-case time complexity bounds are given using the familiar *O*(...) notation. This encourages implementations to use techniques that scale reasonably well and avoid junk algorithms such as bubble sort.

Perhaps a remark about using containers from a multitasking program would be helpful. The general rule is given in paragraph 3 of Annex A which says "The implementation shall ensure that each language defined subprogram is reentrant in the sense that concurrent calls on the same subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects." So in other words we have to protect ourselves by using the normal techniques such as protected objects when container operations are invoked concurrently on the same object from multiple tasks even if the operations are only reading from the container.

2 Lists and vectors

We will first consider the list container since in some ways it is the simplest. Here is its specification interspersed with some explanation

```
generic
   type Element_Type is private;
   with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Doubly_Linked_Lists is
   pragma Preelaborate(Doubly_Linked_Lists);

   type List is tagged private;
   pragma Preelaborable_Initialization(List);
   type Cursor is private;
   pragma Preelaborable_Initialization(Cursor);
   Empty_List: constant List;
   No_Element: constant Cursor;
```

The two generic parameters are the type of the elements in the list and the definition of equality for comparing elements. This equality relation must be such that x = y and y = x always have the same value.

A list container is an object of the type List. It is tagged since it will inevitably be implemented as a controlled type.

The fact that it is visibly tagged means that all the advantages of object oriented programming are available. For one thing it enables the use of the prefixed notation so that we can write operations such as

```
My_List.Append(Some_Value);
```

rather than

```
Append(My_List, Some_Value);
```

The type Cursor is an important concept. It provides the means of access to individual elements in the container. Not only does it contain a reference to an element but it also identifies the container as well. This enables various checks to be made to ensure that we don't accidentally meddle with an element in the wrong container.

The constants Empty_List and No_Element are as expected and also provide default values for objects of types List and Cursor respectively.

```
function "=" (Left, Right: List) return Boolean;
function Length(Container: List) return Count_Type;
function Is_Empty(Container: List) return Boolean;
procedure Clear(Container: in out List);
```

The function "=" compares two lists. It only returns true if both lists have the same number of elements and corresponding elements have the same value as determined by the generic parameter "=" for comparing elements. The subprograms Length, Is_Empty and Clear are as expected.

Note that A_List = Empty_List, Is_Empty(A_List) and Length(A_List) = 0 all have the same value.

```
function Element(Position: Cursor) return Element_Type;

procedure Replace_Element(Container: in out List;
                          Position: in Cursor;
                          New_Item: in Element_Type);
```

These are the first operations we have met that use a cursor. The function Element takes a cursor and returns the value of the corresponding element (remember that a cursor identifies the list as well as the element itself). The procedure Replace_Element replaces the value of the element identified by the cursor by the value given; it makes a copy of course.

Note carefully that Replace_Element has both the list and cursor as parameters. There are two reasons for this concerning correctness. One is to enable a check that the cursor does indeed identify an element in the given list. The other is to ensure that we do have write access to the container (the parameter has mode **in out**). Otherwise it would be possible to modify a container even though we only had a constant view of it. So as a general principle any operation that modifies a container must have the container as a parameter whereas an operation that only reads it such as the function Element does not.

```
procedure Query_Element(Position: in Cursor;
                        Process: not null access procedure
                                 (Element: in Element_Type));

procedure Update_Element(Container: in out List;
                         Position: in Cursor;
                         Process: not null access procedure
                                  (Element: in out Element_Type));
```

These procedures provide *in situ* access to an element. One parameter is the cursor identifying the element and another is an access to a procedure to be called with that element as parameter. In the case of Query_Element, we can only read the element whereas in the case of Update_Element we can change it as well since the parameter mode of the access procedure is **in out**. Note that Update_Element also has the container as a parameter for reasons just mentioned when discussing Replace_Element.

The reader might wonder whether there is any difference between calling the function Element to obtain the current value of an element and using the seemingly elaborate mechanism of Query_Element. The answer is that the function Element makes a copy of the value whereas Query_Element gives access to the value without making a copy. (And similarly for Replace_Element and Update_Element.) This wouldn't matter for a simple list of integers but it would matter if the elements were large or of a controlled type (maybe even lists themselves).

```
procedure Move(Target, Source: in out List);
```

This moves the list from the source to the target after first clearing the target. It does not make copies of the elements so that after the operation the source is empty and Length(Source) is zero.

```
procedure Insert(Container: in out List;
                 Before: in Cursor;
                 New_Item: in Element_Type;
                 Count: in Count_Type := 1);

procedure Insert(Container: in out List;
                 Before: in Cursor;
                 New_Item: in Element_Type;
                 Position: out Cursor;
                 Count: in Count_Type := 1);

procedure Insert(Container: in out List;
                 Before: in Cursor;
                 Position: out Cursor;
                 Count: in Count_Type := 1);
```

These three procedures enable one or more identical elements to be added anywhere in a list. The place is indicated by the parameter Before – if this is No_Element, then the new elements are added at the end. The second procedure is similar to the first but also returns a cursor to the first of the added elements. The third is like the second but the new elements take their default values. Note the default value of one for the number of elements.

```
procedure Prepend(Container: in out List;
                  New_Item: in Element_Type;
                  Count: in Count_Type := 1);

procedure Append(Container: in out List;
                 New_Item: in Element_Type;
                 Count: in Count_Type := 1);
```

These add one or more new elements at the beginning or end of a list respectively. Clearly these operations can be done using Insert but they are sufficiently commonly needed that it is convenient to provide them specially.

```
procedure Delete(Container: in out List;
                 Position: in out Cursor;
                 Count: in Count_Type := 1);

procedure Delete_First(Container: in out List;
                 Count: in Count_Type := 1);

procedure Delete_Last(Container: in out List;
                 Count: in Count_Type := 1);
```

These delete one or more elements at the appropriate position. In the case of Delete, the parameter Position is set to No_Element upon return. If there are not as many as Count elements to be deleted at the appropriate place then it just deletes as many as possible (this clearly results in the container becoming empty in the case of Delete_First and Delete_Last).

```
procedure Reverse_Elements(Container: in out List);
```

This does the obvious thing. It would have been nice to call this procedure Reverse but sadly that is a reserved word.

```
procedure Swap(Container: in out List; I, J: in Cursor);
```

This handy procedure swaps the values in the two elements denoted by the two cursors. The elements must be in the given container otherwise Program_Error is raised. Note that the cursors do not change.

```
procedure Swap_Links(Container: in out List;
                     I, J: in Cursor);
```

This performs the low level operation of swapping the links rather than the values which can be much faster if the elements are large. There is no analogy in the vectors package.

```
procedure Splice(Target: in out List;
                 Before: in Cursor;
                 Source in out List);

procedure Splice(Target: in out List;
                 Before: in Cursor;
                 Source: in out List;
                 Position: in out Cursor);

procedure Splice(Container: in out List;
                 Before: in Cursor;
                 Position: in out Cursor);
```

These three procedures enable elements to be moved (without copying). The place is indicated by the parameter Before – if this is No_Element, then the elements are added at the end. The first moves all the elements of Source into Target at the position given by Before; as a consequence, like the procedure Move, after the operation the source is empty and Length(Source) is zero. The second moves a single element at Position from the list Source to Target and so the length of target is incremented whereas that of source is decremented; Position is updated to its new location in Target. The third moves a single element within a list and so the length remains the same (note the formal parameter is Container rather than Target in this case). There are no corresponding operations in the vectors package because, like Swap_Links, we are just moving the links and not copying the elements.

```
function First(Container: List) return Cursor;
function First_Element(Container: List)
                                  return Element_Type;
function Last(Container: List) return Cursor;
function Last_Element(Container: List)
                                  return Element_Type;
function Next(Position: Cursor) return Cursor;
function Previous(Position: Cursor) return Cursor;
procedure Next(Position: in out Cursor);
procedure Previous(Position: in out Cursor);
function Find(Container: List;
              Item: Element_Type;
              Position: Cursor:= No_Element)
                                  return Cursor;
function Reverse_Find(Container: List;
              Item: Element_Type;
              Position: Cursor:= No_Element)
                                  return Cursor;
function Contains(Container: List;
                  Item: Element_Type) return Boolean;
```

Hopefully the purpose of these is almost self-evident. The function Find searches for an element with the given value starting at the given cursor position (or at the beginning if the position is No_Element); if no element is found then it returns No_Element. Reverse_Find does the same but backwards. Note that equality used for the comparison in Find and Reverse_Find is that defined by the generic parameter "=".

```
function Has_Element(Position: Cursor) return Boolean;
```

This returns False if the cursor does not identify an element; for example if it is No_Element.

```
procedure Iterate(Container: in List;
          Process: not null access procedure
                                  (Position: in Cursor));

procedure Reverse_Iterate(Container: in List;
          Process: not null access procedure
                                  (Position: in Cursor));
```

These apply the procedure designated by the parameter Process to each element of the container in turn in the appropriate order.

```
generic
  with function "<" (Left, Right: Element_Type)
                                  return Boolean is <>;
package Generic_Sorting is
  function Is_Sorted(Container: List) return Boolean;
  procedure Sort(Container: in out List);
  procedure Merge(Target, Source: in out List);
end Generic_Sorting;
```

This generic package performs sort and merge operations using the order specified by the generic formal parameter.

Note that we use generics rather than access to subprogram parameters when the formal process is given by an operator. This is because the predefined operations have convention Intrinsic and one cannot pass an intrinsic operation as an access to subprogram parameter. The function Is_Sorted returns True if the container is already sorted. The procedure Sort arranges the elements into order as necessary – note that no copying is involved since it is only the links that are moved. The procedure Merge takes the elements from Source and adds them to Target. After the merge Length(Source) is zero. If both lists were sorted before the merge then the result is also sorted.

And finally we have

```
private
   ... -- not specified by the language
end Ada.Containers.Doubly_Linked_Lists;
```

If the reader has got this far they have probably understood how to use this package so extensive examples are unnecessary. However, as a taste, here is a simple stack of floating point numbers

```
package Stack is
   procedure Push(X: in Float);
   function Pop return Float;
   function Size return Integer;
   exception Stack_Empty;
end;

with Ada.Containers.Doubly_Linked_Lists;
use Ada.Containers;
package body Stack is

   package Float_Container is
                  new Doubly_Linked_Lists(Float);
   use Float_Container;
   The_Stack: List;

   procedure Push(X: in Float) is
   begin
      Append(The_Stack, X);   -- or The_Stack.Append(X);
   end Push;

   function Pop return Float is
      Result: Float;
   begin
      if Is_Empty(The_Stack) then
         raise Stack_Empty;
      end if;
      Result := Last_Element(The_Stack);
      Delete_Last(The_Stack);
      return Result;
   end Pop;

   function Size return Integer is
   begin
      return Integer(Length(The_Stack));
   end Size;
end Stack;
```

This barely needs any explanation. The lists package is instantiated in the package Stack and the object The_Stack is of course the list container. The rest is really straightforward. We could of course use the prefixed notation throughout as indicated in Push.

An important point should be mentioned concerning lists (and containers in general). This is that attempts to do foolish things typically result in Constraint_Error or Program_Error being raised. This especially applies to the procedures Process in Query_Element, Update_Element, Iterate and Reverse_Iterate. The concepts of tampering with cursors and elements are introduced in order to dignify a general motto of "Thou shalt not violate thy container".

Tampering with cursors occurs when elements are added to or deleted from a container (by calling Insert and so on) whereas tampering with elements means replacing an element (by calling Replace_Element for example). Tampering with elements is a greater sin and includes tampering with cursors. The procedure Process in Query_Element and Update_Element must not tamper with elements and the procedure Process in the other cases must not tamper with cursors. The reader might think it rather odd that Update_Element should not be allowed to tamper with elements since the whole purpose is to update the element; this comes back to the point mentioned earlier that update element gives access to the existing element *in situ* via the parameter of Process and that is allowed – calling Replace_Element within Process would be tampering. Tampering causes Program_Error to be raised.

We will now consider the vectors package. Its specification starts

```
generic
   type Index_Type is range <>;
   type Element_Type is private;
   with function "=" (Left, Right: Element_Type)
                                 return Boolean is <>;
package Ada.Containers.Vectors is
   pragma Preelaborate(Vectors);
```

This is similar to the lists package except for the additional generic parameter Index_Type (note that this is an integer type and not a discrete type). This additional parameter reflects the idea that a vector is essentially an array and we can index directly into an array.

In fact the vectors package enables us to access elements either by using an index or by using a cursor. Thus many operations are duplicated such as

```
function Element(Container: Vector; Index: Index_Type)
                                       return Element_Type;
function Element(Position: Cursor) return Element_Type;
procedure Replace_Element(Container: in out Vector;
                          Index: in Index_Type;
                          New_Item: in Element_Type);
procedure Replace_Element(Container: in out Vector;
                          Position: in Cursor;
                          New_Item: in Element_Type);
```

If we use an index then there is always a distinct parameter identifying the vector as well. If we use a cursor then the vector parameter is omitted if the vector is unchanged as is the case with the function Element. Remember that we

stated earlier that a cursor identifies both an element and the container but if the container is being changed as in the case of Replace_Element then the container has to be passed as well to ensure write access and to enable a check that the cursor does identify an element in the correct container.

There are also functions First_Index and Last_Index thus

```
function First_Index(Container: Vector)
                                    return Index_Type;
function Last_Index(Container: Vector)
                                    return Extended_Index;
```

These return the values of the index of the first and last elements respectively. The function First_Index always returns Index_Type'First whereas Last_Index will return No_Index if the vector is empty. The function Length returns Last_Index–First_Index+1 which is zero if the vector is empty. Note that the irritating subtype Extended_Index has to be introduced in order to cope with end values. The constant No_Index has the value Extended_Index'First which is equal to Index_Type'First–1.

There are operations to convert between an index and a cursor thus

```
function To_Cursor(Container: Vector;
                   Index: Extended_Index) return Cursor;
function To_Index(Position: Cursor)
                                    return Extended_Index;
```

It is perhaps slightly messier to use the index and vector parameters because of questions concerning the range of values of the index but probably slightly faster and maybe more familiar. And sometimes of course using an index is the whole essence of the problem. In the paper on access types we showed a use of the procedure Update_Element to double the values of those elements of a vector whose index was in the range 5 to 10. This would be tedious with cursors.

But an advantage of using cursors is that (provided certain operations are avoided) it is easy to replace the use of vectors by lists.

For example here is the stack package rewritten to use vectors

```
with Ada.Containers.Vectors;                -- changed
use Ada.Containers;
package body Stack is

  package Float_Container is
            new Vectors(Natural, Float);    -- changed
  use Float_Container;
  The_Stack: Vector;                        -- changed

  procedure Push(X: in Float) is
  begin
    Append(The_Stack, X);
  end Push;

  -- etc exactly as before

end Stack;
```

So the changes are very few indeed and can be quickly done with a simple edit.

Note that the index parameter has been given as Natural rather than Integer. Using Integer will not work since attempting to elaborate the subtype Extended_Index would raise Constraint_Error when evaluating Integer'First–1. But in any event it is more natural for the index range of the container to start at 0 (or 1) rather than a large negative value such as Integer'First.

There are other important properties of vectors that should be mentioned. One is that there is a concept of capacity. Vectors are adjustable and will extend if necessary when new items are added. However, this might lead to lots of extensions and copying and so we can set the capacity of a container by calling

```
procedure Reserve_Capacity(Container: in out Vector;
                           Capacity: in Count_Type);
```

There is also

```
function Capacity(Container: Vector) return Count_Type;
```

which naturally returns the current capacity. Note that Length(V) cannot exceed Capacity(V) but might be much less.

If we add items to a vector whose length and capacity are the same then no harm is done. The capacity will be expanded automatically by effectively calling Reserve_Capacity internally. So the user does not need to set the capacity although not doing so might result in poorer performance.

There is also the concept of "empty elements". These are elements whose values have not been set. There is no corresponding concept with lists. It is a bounded error to read an empty element. Empty elements arise if we declare a vector by calling

```
function To_Vector(Length: Count_Type) return Vector;
```

as in

```
My_Vector: Vector := To_Vector(100);
```

There is also the much safer

```
function To_Vector(New_Item: Element_Type;
                   Length: Count_Type) return Vector;
```

which sets all the elements to the value New_Item.

There is also a procedure

```
procedure Set_Length(Container: in out Vector;
                     Length: in Count_Type);
```

This changes the length of a vector. This may require elements to be deleted (from the end) or to be added (in which case the new ones are empty).

The final way to get an empty element is by calling one of

```
procedure Insert_Space(Container: in out Vector;
                       Before: in Extended_Index;
                       Count: in Count_Type := 1);
```

```
procedure Insert_Space(Container: in out Vector;
                       Before: in Cursor;
                       Position: out Cursor;
                       Count: in Count_Type := 1);
```

These insert the number of empty elements given by Count at the place indicated. Existing elements are slid along as necessary. These should not be confused with the versions of Insert which do not provide an explicit value for the elements – in those cases the new elements take their default values.

Care needs to be taken if we use empty elements. For example we should not compare two vectors using "=" if they have empty elements because this implies reading them. But the big advantage of empty elements is that they provide a quick way to make a large lump of space in a vector which can then be filled in with appropriate values. One big slide is a lot faster than lots of little ones.

For completeness, we briefly mention the remaining few subprograms that are unique to the vectors package.

There are further versions of Insert thus

```
procedure Insert(Container: in out Vector;
      Before: in Extended_Index; New_Item: in Vector);

procedure Insert(Container: in out Vector;
      Before: in Cursor; New_Item: in Vector);

procedure Insert(Container: in out Vector;
      Before: in Cursor; New_Item: in Vector;
                         Position: out Cursor);
```

These insert copies of a vector into another vector (rather than just single elements).

There are also corresponding versions of Prepend and Append thus

```
procedure Prepend(Container: in out Vector;
                  New_Item: in Vector);

procedure Append(Container: in out Vector;
                 New_Item: in Vector);
```

Finally, there are four functions "&" which concatenate vectors and elements by analogy with those for the type String. Their specifications are

```
function "&" (Left, Right: Vector) return Vector;
function "&" (Left: Vector; Right: Element_Type)
                                        return Vector;
function "&" (Left: Element_Type; Right: Vector)
                                        return Vector;
function "&" (Left, Right: Element_Type) return Vector;
```

Note the similarity between

```
Append(V1, V2);
V1 := V1 & V2;
```

The result is the same but using "&" is less efficient because of the extra copying involved. But "&" is a familiar operation and so is provided for convenience.

3 Maps

We will now turn to the maps and sets packages. We will start by considering maps which are more exciting than sets and begin with ordered maps which are a little simpler and then consider hashed maps.

Remember that a map is just a means of getting from a value of one type (the key) to another type (the element). This is not a one-one relationship. Given a key there is a unique element (if any), but several keys may correspond to the same element. A simple example is an array. This is a map from the index type to the component type. Thus if we have

```
S: String := "animal";
```

then this provides a map from integers in the range 1 to 6 to some values of the type Character. Given an integer such as 3 there is a unique character 'i' but given a character such as 'a' there might be several corresponding integers (in this case both 1 and 5).

More interesting examples are where the set of used key values is quite sparse. For example we might have a store where various spare parts are held. The parts have a five-digit part number and there are perhaps twenty racks where they are held identified by a letter. However, only a handful of the five digit numbers are in use so it would be very wasteful to use an array with the part number as index. What we want instead is a container which holds just the pairs that matter such as (34618, 'F'), (27134, 'C') and so on. We can do this using a map. We usually refer to the pairs of values as nodes of the map.

There are two maps packages with much in common. One keeps the keys in order and the other uses a hash function. Here is the specification of the ordered maps package generally showing just those facilities common to both.

```
generic
  type Key_Type is private;
  type Element_Type is private;
  with function "<" (Left, Right: Key_Type)
                                  return Boolean is <>;
  with function "=" (Left, Right: Element_Type)
                                  return Boolean is <>;
package Ada.Containers.Ordered_Maps is
  pragma Preelaborate(Ordered_Maps);

  function Equivalent_Keys(Left: Right: Key_Type)
                                        return Boolean;
```

The generic parameters include the ordering relationship "<" on the keys and equality for the elements.

It is assumed that the ordering relationship is well behaved in the sense that if x < y is true then y < x is false. We say that two keys x and y are equivalent if both x < y and y < x are false. In other words this defines an equivalence class on keys. The relationship must also be transitive, that is, if x < y and y < z are both true then x < z must also be true.

This concept of an equivalence relationship occurs throughout the various maps and sets. Sometimes, as here,

it is defined in terms of an order but in other cases, as we shall see, it is defined by an equivalence function.

It is absolutely vital that the equivalence relations are defined properly and meet the above requirements. It is not possible for the container packages to check this and if the operations are wrong then peculiar behaviour is almost inevitable.

For the convenience of the user the function Equivalent_Keys is declared explicitly. It is equivalent to

```
function Equivalent_Keys(Left, Right: Key_Type)
                                    return Boolean is
begin
  return not(Left < Right) and not(Right < Left);
end Equivalent_Keys;
```

The equality operation on elements is not so demanding. It must be symmetric so that x = y and y = x are the same but transitivity is not required (although cases where it would not automatically be transitive are likely to be rare). The operation is only used for the function "=" on the containers as a whole.

Note that Find and similar operations for maps and sets work in terms of the equivalence relationship rather than equality as was the case with lists and vectors.

```
type Map is tagged private;
pragma Preelaborable_Initialization(Map);
type Cursor is private;
pragma Preelaborable_Initialization(Cursor);
Empty_Map: constant Map;
No_Element: constant Cursor;
```

The types Map and Cursor and constants Empty_Map and No_Element are similar to the corresponding entities in the lists and vectors containers.

```
function "=" (Left, Right: Map) return Boolean;
function Length(Container: Map) return Count_Type;
function Is_Empty(Container: Map) return Boolean;
procedure Clear(Container: in out Map);
```

These are again similar to the corresponding entities for lists. Note that two maps are said to be equal if they have the same number of nodes with equivalent keys (as defined by "<") whose corresponding elements are equal (as defined by "=").

```
function Key(Position: Cursor) return Key_Type;
function Element(Position: Cursor) return Element_Type;

procedure Replace_Element(Container: in out Map;
                          Position: in Cursor;
                          New_Item: in Element_Type);

procedure Query_Element(Position: in Cursor;
                Process: not null access procedure
                          (Key: in Key_Type;
                              Element: in Element_Type));

procedure Update_Element(Container: in out Map;
                Position: in Cursor;
                Process: not null access procedure
                          (Key: in Key_Type;
                          Element: in out Element_Type));
```

In this case there is a function Key as well as a function Element. But there is no procedure Replace_Key since it would not make sense to change a key without changing the element as well and this really comes down to deleting the whole node and then inserting a new one.

The procedures Query_Element and Update_Element are slightly different in that the procedure Process also takes the key as parameter as well as the element to be read or updated. Note again that the key cannot be changed. Nevertheless the value of the key is given since it might be useful in deciding how the update should be performed. Remember that we cannot get uniquely from an element to a key but only from a key to an element.

```
procedure Move(Target, Source: in out Map);
```

This moves the map from the source to the target after first clearing the target. It does not make copies of the nodes so that after the operation the source is empty and Length(Source) is zero.

```
procedure Insert(Container: in out Map;
                 Key: in Key_Type;
                 New_Item: in Element_Type;
                 Position: out Cursor;
                 Inserted: out Boolean);

procedure Insert(Container: in out Map;
                 Key: in Key_Type;
                 Position: out Cursor;
                 Inserted: out Boolean);

procedure Insert(Container: in out Map;
                 Key: in Key_Type;
                 New_Item: in Element_Type);
```

These insert a new node into the map unless a node with an equivalent key already exists. If it does exist then the first two return with Inserted set to False and Position indicating the node whereas the third raises Constraint_Error (the element value is not changed). If a node with equivalent key is not found then a new node is created with the given key, the element value is set to New_Item when that is given and otherwise it takes its default value (if any), and Position is set when given.

Unlike vectors and lists, we do not have to say where the new node is to be inserted because of course this is an ordered map and it just goes in the correct place according to the order given by the generic parameter "<".

```
procedure Include(Container: in out Map;
                 Key: in Key_Type;
                 New_Item: in Element_Type);
```

This is somewhat like the last Insert except that if an existing node with an equivalent key is found then it is replaced (rather than raising Constraint_Error). Note that both the key and the element are updated. This is because equivalent keys might not be totally equal.

For example the key part might be a record with part number and year of introduction, thus

```
type Part_Key is
  record
    Part_Number: Integer;
    Year: Integer;
  end record;
```

and we might define the ordering relationship to be used as the generic parameter simply in terms of the part number

```
function "<" (Left, Right: Part_Key) return Boolean is
begin
  return Left.Part_Number < Right.Part_Number;
end "<";
```

In this situation, the keys could match without the year component being the same and so it would need to be updated. In other words with this definition of the ordering relation, two keys are equivalent provided just the part numbers are the same.

```
procedure Replace(Container: in out Map;
                  Key: in Key_Type;
                  New_Item: in Element_Type);
```

In this case, Constraint_Error is raised if the node does not already exist. On replacement both the key and the element are updated as for Include.

Perhaps a better example of equivalent keys not being totally equal is if the key were a string. We might decide that the case of letter did not need to match in the test for equivalence but nevertheless we would probably want to update with the string as used in the parameter of Replace.

```
procedure Exclude(Container: in out Map;
                  Key: in Key_Type);
```

If there is a node with an equivalent key then it is deleted. If there is not then nothing happens.

```
procedure Delete(Container: in out Map;
                 Key: in Key_Type);

procedure Delete(Container: in out Map;
                 Position: in out Cursor);
```

These delete a node. In the first case if there is no such equivalent key then Constraint_Error is raised (by contrast to Exclude which remains silent in this case). In the second case if the cursor is No_Element then again Constraint_Error is raised – there is also a check to ensure that the cursor otherwise does designate a node in the correct map (remember that cursors designate both an entity and the container); if this check fails then Program_Error is raised.

Perhaps it is worth observing that Insert, Include, Replace, Exclude and Delete form a sort of progression from an operation that will insert something, through operations that might insert, will neither insert nor delete, might delete, to the final operation that will delete something. Note also that Include, Replace and Exclude do not apply to lists and vectors.

```
function First(Container: Map) return Cursor;
function Last(Container: Map) return Cursor;
function Next(Position: Cursor) return Cursor;
procedure Next(Position: in out Cursor);
function Find(Container: Map;
              Key: Key_Type) return Cursor;
function Element(Container: Map;
                 Key: Key_Type) return Element;
function Contains(Container: Map;
                  Key: Key_Type) return Boolean;
```

These should be self-evident. Unlike the operations on vectors and lists, Find logically searches the whole map and not just starting at some point (and since it searches the whole map there is no point in having Reverse_Find). (In implementation terms it won't actually search the whole map because it will be structured in a way that makes this unnecessary – as a balanced tree perhaps.) Moreover, Find uses the equivalence relation based on the "<" parameter so in the example it only has to match the part number and not the year. The function call Element(My_Map, My_Key) is equivalent to Element(Find(My_Map, My_Key)).

```
function Has_Element(Position: Cursor) return Boolean;

procedure Iterate(Container: in Map;
                  Process: not null access procedure
                                (Position: in Cursor));
```

These are also as for other containers.

And at last we have

```
private
  ... -- not specified by the language
end Ada.Containers.Ordered_Maps;
```

We have omitted to mention quite a few operations that have no equivalent in hashed maps – we will come back to these in a moment.

As an example we can make a container to hold the information concerning spare parts. We can use the type Part_Key and the function "<" as above. We can suppose that the element type is

```
type Stock_Info is
  record
    Shelf: Character range 'A' .. 'T';
    Stock: Integer;
  end record;
```

This gives both the shelf letter and the number in stock.

We can then declare the container thus

```
package Store_Maps is
      new Ordered_Maps(Key_Type => Part_Key,
                       Element_Type => Stock_Info,
                       "<" => "<");

The_Store: Store_Maps.Map;
```

The last parameter could be omitted because the formal has a <> default.

We can now add items to our store by calling

```
The_Store.Insert((34618, 1998), ('F', 25));
The_Store.Insert((27134, 2004), ('C', 45));
...
```

We might now have a procedure which, given a part number, checks to see if it exists and that the stock is not zero, and if so returns the shelf letter and year number and decrements the stock count.

```
procedure Request(Part: in Integer; OK: out Boolean;
                  Year: out Integer; Shelf: out Character) is
  C: Cursor;
  K: Part_Key;
  E: Stock_Info;
begin
  C := The_Store.Find((Part, 0));
  if C = No_Element then
    OK := False; return;                    -- no such key
  end if;
  E := Element(C);  K := Key(C);
  Year := K.Year;  Shelf := E.Shelf;
  if E.Stock = 0 then
    OK := False; return;                    -- out of stock
  end if;
  Replace_Element(C, (Shelf, E.Stock–1));
  OK := True;
end Request;
```

Note that we had to put a dummy year number in the call of Find. We could of course use the new <> notation for this

```
C := The_Store.Find((Part, others => <>));
```

The reader can improve this example at leisure – by using Update_Element for example.

As another example suppose we wish to check all through the stock looking for parts whose stock is low, perhaps less than some given parameter. We can use Iterate for this as follows

```
procedure Check_Stock(Low: in Integer) is

  procedure Check_It(C: in Cursor) is
  begin
    if Element(C).Stock < Low then
      -- print a message perhaps
      Put("Low stock of part ");
      Put_Line(Key(C).Part_Number);
    end if;
  end Check_It;

begin
  The_Store.Iterate(Check_It'Access);
end Check_Stock;
```

Note that this uses a so-called downward closure. The procedure Check_It has to be declared locally to Check_Stock in order to access the parameter Low. (Well you could declare it outside and copy the parameter Low to a global variable but that is just the sort of wicked thing one has to do in lesser languages (such as even Ada 95). It is not task safe for one thing.)

Another approach is to use First and Next and so on thus

```
procedure Check_Stock(Low: in Integer) is
  C: Cursor := The_Store.First;
begin
  loop
    exit when C = No_Element;
    if Element(C).Stock < Low then
      -- print a message perhaps
      Put("Low stock of part ");
      Put_Line(Key(C).Part_Number);
    end if;
    C := The_Store.Next(C);
  end loop;
end Check_Stock;
```

We will now consider hashed maps. The trouble with ordered maps in general is that searching can be slow when the map has many entries. Techniques such as a binary tree can be used but even so the search time will increase at least as the logarithm of the number of entries. A better approach is to use a hash function. This will be familiar to many readers (especially those who have written compilers). The general idea is as follows.

We define a function which takes a key and returns some value in a given range. In the case of the Ada containers it has to return a value of the modular type Hash_Type which is declared in the root package Ada.Containers. We could then convert this value onto a range representing an index into an array whose size corresponds to the capacity of the map. This index value is the preferred place to store the entry. If there already is an entry at this place (because some other key has hashed to the same value) then a number of approaches are possible. One way is to create a list of entries with the same index value (often called buckets); another way is simply to put it in the next available slot. The details don't matter. But the overall effect is that provided the map is not too full and the hash function is good then we can find an entry almost immediately more or less irrespective of the size of the map.

So as users all we have to do is to define a suitable hash function. It should give a good spread of values across the range of Hash_Type for the population of keys, it should avoid clustering and above all for a given key it must *always* return the same hash value. A good discussion on hash functions by Knuth will be found in [1].

Defining good hash functions needs care. In the case of the part numbers we might multiply the part number by some obscure prime number and then truncate the result down to the modular type Hash_Type. The author hesitates to give an example but perhaps

```
function Part_Hash(P: Part_Key) return Hash_Type is
  M31: constant := 2**31–1;     -- a nice Mersenne prime
begin
  return Hash_Type(P.Part_Number) * M31;
end Part_Hash;
```

On reflection that's probably a very bad prime to use because it is so close to half of 2**32 a typical value of Hash_Type'Last+1. Of course it doesn't have to be prime

but simply relatively prime to it such as 5**13. Knuth suggests dividing the range by the golden number $\tau = (\sqrt{5}+1)/2 = 1.618...$ and then taking the nearest number relatively prime which is in fact simply the nearest odd number (in this case it is 2654435769).

Here is a historic interlude. Marin Mersenne (1588-1648) was a Franciscan monk who lived in Paris. He studied numbers of the form $M_p = 2^p - 1$ where p is prime. A lot of these are themselves prime. Mersenne gave a list of those upto 257 which he said were prime (namely 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257). It was not until 1947 that it was finally settled that he got some of them wrong (61, 89, and 107 are also prime but 67 and 257 are not). At the time of writing there are 42 known Mersenne primes and the largest which is also the largest known prime number is $M_{25964951}$ – see www.mersenne.org.

The specification of the hashed maps package is very similar to that for ordered maps. It starts

```
generic
   type Key_Type is private;
   type Element_Type is private;
   with function Hash(Key: Key_Type) return Hash_Type;
   with function Equivalent_Keys(Left, Right: Key_Type)
                                    return Boolean;
   with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Hashed_Maps is
   pragma Preelaborate(Hashed_Maps);
```

The differences from the ordered maps package are that there is an extra generic parameter Hash giving the hash function and the ordering parameter "<" has been replaced by the function Equivalent_Keys. It is this function that defines the equivalence relationship for hashed maps; it is important that Equivalent_Keys(X, Y) is always the same as Equivalent_Keys(Y, X). Moreover if X and Y are equivalent and Y and Z are equivalent then X and Z must also be equivalent.

Note that the function Equivalent_Keys in the ordered maps package discussed above corresponds to the formal generic parameter of the same name in this hashed maps package. This should make it easier to convert between the two forms of packages.

Returning to our example, if we now write

```
function Equivalent_Parts(Left, Right: Part_Key)
                                    return Boolean is
begin
   return Left.Part_Number = Right.Part_Number;
end Equivalent_Parts;
```

then we can instantiate the hashed maps package as follows

```
package Store_Maps is
      new Hashed_Maps(Key_Type => Part_Key,
                  Element_Type => Stock_Info,
                  Hash => Part_Hash,
                  Equivalent_Keys => Equivalent_Parts);

The_Store: Store_Maps.Map;
```

and then the rest of our example will be exactly as before. It is thus easy to convert from an ordered map to a hashed map and vice versa provided of course that we only use the facilities common to both.

We will finish this discussion of maps by briefly considering the additional facilities in the two packages.

The ordered maps package has the following additional subprograms

```
procedure Delete_First(Container: in out Map);
procedure Delete_Last(Container: in out Map);
function First_Element(Container: Map)
                                    return Element_Type;
function First_Key(Container: Map) return Key_Type;
function Last_Element(Container: Map)
                                    return Element_Type;
function Last_Key(Container: Map) return Key_Type;
function Previous(Position: Cursor) return Cursor;
procedure Previous(Position: in out Cursor);
function Floor(Container: Map;
                          Key: Key_Type) return Cursor;
function Ceiling(Container: Map;
                          Key: Key_Type) return Cursor;
function "<" (Left, Right: Cursor) return Boolean;
function ">" (Left, Right: Cursor) return Boolean;
function "<" (Left: Cursor; Right: Key_Type)
                                    return Boolean;
function ">" (Left: Cursor; Right: Key_Type)
                                    return Boolean;
function "<" (Left: Key_Type; Right: Cursor)
                                    return Boolean;
function ">" (Left: Key_Type; Right: Cursor)
                                    return Boolean;
procedure Reverse_Iterate(Container: in Map;
                Process: not null access procedure
                                (Position: in Cursor));
```

These are again largely self-evident. The functions Floor and Ceiling are interesting. Floor searches for the last node whose key is not greater than Key and similarly Ceiling searches for the first node whose key is not less than Key – they return No_Element if there is no such element. The subprograms Previous are of course the opposite of Next and Reverse_Iterate is like Iterate only backwards.

The functions "<" and ">" are mostly for convenience. Thus the first is equivalent to

```
function "<" (Left, Right: Cursor) return Boolean is
begin
   return Key(Left) < Key(Right);
end "<";
```

Clearly these additional operations must be avoided if we wish to retain the option of converting to a hashed map later.

Hashed maps have a very important facility not in ordered maps which is the ability to specify a capacity as for the vectors package. (Underneath their skin the hashed maps are a bit like vectors whereas the ordered maps are a bit like lists.) Thus we have

```
procedure Reserve_Capacity(Container: in out Map;
                           Capacity: in Count_Type);

function Capacity(Container: Map) return Count_Type;
```

The behaviour is much as for vectors. We don't have to set the capacity ourselves since it will be automatically extended as necessary but it might significantly improve performance to do so. In the case of maps, increasing the capacity requires the hashing to be redone which could be quite time consuming, so if we know that our map is going to be a big one, it is a good idea to set an appropriate capacity right from the beginning. Note again that Length(M) cannot exceed Capacity(M) but might be much less.

The other additional subprograms for hashed maps are

```
function Equivalent_Keys(Left, Right: Cursor)
                                        return Boolean;
function Equivalent_Keys(Left: Cursor;
                         Right: Key_Type) return Boolean;
function Equivalent_Keys(Left: Key_Type;
                         Right: Cursor) return Boolean;
```

These (like the additional "<" and ">" for ordered maps) are again mostly for convenience. The first is equivalent to

```
function Equivalent_Keys(Left, Right: Cursor)
                                        return Boolean is
begin
  return Equivalent_Keys(Key(Left), Key(Right));
end Equivalent_Keys;
```

Before moving on to sets it should be noticed that there are also some useful functions in the string packages. The main one is

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

There is a similar function Ada.Strings.Unbounded.Hash where the parameter Key has type Unbounded_String. It simply converts the parameter to the type String and then calls Ada.Strings.Hash. There is also a generic function for bounded strings which again calls the basic function Ada.Strings.Hash. For completeness the function Ada.Strings.Fixed.Hash is a renaming of Ada.Strings.Hash.

These are provided because it is often the case that the key is a string and they save the user from devising good hash functions for strings which might cause a nasty headache.

We could for example save ourselves the worry of defining a good hash function in the above example by making the part number into a 5-character string. So we might write

```
function Part_Hash(P: Part_Key) return Hash_Type is
begin
  return Ada.Strings.Hash(P.Part_Number);
end Part_Hash;
```

and if this doesn't work well then we can blame the vendor.

4 Sets

Sets, like maps, come in two forms: hashed and ordered. Sets are of course just collections of values and there is no question of a key (we can perhaps think of the value as being its own key). Thus in the case of an ordered set the values are stored in order whereas in the case of a map, it is the keys that are stored in order. As well as the usual operations of inserting elements into a set and searching and so on, there are also many operations on sets as a whole that do not apply to the other containers – these are the familiar set operations such as union and intersection.

Here is the specification of the ordered sets package giving just those facilities that are common to both kinds of sets.

```
generic
  type Element_Type is private;
  with function "<" (Left, Right: Element_Type)
                                        return Boolean is <>;
  with function "=" (Left, Right: Element_Type)
                                        return Boolean is <>;
package Ada.Containers.Ordered_Sets is
  pragma Preelaborate(Ordered_Sets);

  function Equivalent_Elements(Left, Right:
                               Element_Type) return Boolean;

  type Set is tagged private;
  pragma Preelaborable_Initialization(Set);
  type Cursor is private;
  pragma Preelaborable_Initialization(Cursor);
  Empty_Set: constant Set;
  No_Element: constant Cursor;
```

The only differences from the maps package (apart from the identifiers) are that there is no key type and both "<" and "=" apply to the element type (whereas in the case of maps, the operation "<" applies to the key type). Thus the ordering relationship "<" defined on elements defines equivalence between the elements whereas "=" defines equality.

It is possible for two elements to be equivalent but not equal. For example if they were strings then we might decide that the ordering (and thus equivalence) ignored the case of letters but that equality should take the case into account. (They could also be equal but not equivalent but that is perhaps less likely.)

And as in the case of the maps package, the equality operation on elements is only used by the function "=" for comparing two sets.

Again we have the usual rules as explained for maps. Thus if x < y is true then y < x must be false; x < y and y < z must imply x < z; and x = y and y = x must be the same.

For the convenience of the user the function Equivalent_Elements is declared explicitly. It is equivalent to

```
function Equivalent_Elements(Left, Right:
                             Element_Type) return Boolean is
begin
  return not(Left < Right) and not(Right < Left);
end Equivalent_Elements;
```

This function Equivalent_Elements corresponds to the formal generic parameter of the same name in the hashed sets package discussed below. This should make it easier to convert between the two forms of packages.

```
function "=" (Left, Right: Set) return Boolean;
function Equivalent_Sets(Left, Right: Set) return Boolean;
function To_Set(New_Item: Element_Type) return Set;
function Length(Container: Set) return Count_Type;
function Is_Empty(Container: Set) return Boolean;
procedure Clear(Container: in out Set);
```

Note the addition of Equivalent_Sets and To_Set. Two sets are equivalent if they have the same number of elements and the pairs of elements are equivalent. This contrasts with the function "=" where the pairs of elements have to be equal rather than equivalent. Remember that elements might be equivalent but not equal (as in the example of a string mentioned above). The function To_Set takes a single element and creates a set. It is particularly convenient when used in conjunction with operations such as Union described below. The other subprograms are as in the other containers.

```
function Element(Position: Cursor) return Element_Type;

procedure Replace_Element(Container: in out Set;
                          Position: in Cursor;
                          New_Item: in Element_Type);

procedure Query_Element(Position: in Cursor;
                        Process: not null access procedure
                                 (Element: in Element_Type));
```

Again these are much as expected except that there is no procedure Update_Element. This is because the elements are arranged in terms of their own value (either by order or through the hash function) and if we just change an element *in situ* then it might become out of place (this problem does not arise with the other containers). This also means that Replace_Element has to ensure that the value New_Item is not equivalent to an element in a different position; if it is then Program_Error is raised. We will return to the problem of the missing Update_Element later.

```
procedure Move(Target, Source: in out Set);
```

This is just as for the other containers.

```
procedure Insert(Container: in out Set;
                 New_Item: in Element_Type;
                 Position: out Cursor;
                 Inserted: out Boolean);

procedure Insert(Container: in out Set;
                 New_Item: in Element_Type);
```

These insert a new element into the set unless an equivalent element already exists. If it does exist then the first one returns with Inserted set to False and Position indicating the element whereas the second raises Constraint_Error (the element value is not changed). If an equivalent element is not in the set then it is added and Position is set accordingly.

```
procedure Include(Container: in out Set;
                  New_Item: in Element_Type);
```

This is somewhat like the last Insert except that if an equivalent element is already in the set then it is replaced (rather than raising Constraint_Error).

```
procedure Replace(Container: in out Set;
                  New_Item: in Element_Type);
```

In this case, Constraint_Error is raised if an equivalent element does not already exist.

```
procedure Exclude(Container: in out Set;
                  Item: in Element_Type);
```

If an element equivalent to Item is already in the set, then it is deleted.

```
procedure Delete(Container: in out Set;
                 Item: in Element_Type);

procedure Delete(Container: in out Set;
                 Position: in out Cursor);
```

These delete an element. In the first case if there is no such equivalent element then Constraint_Error is raised. In the second case if the cursor is No_Element then again Constraint_Error is also raised – there is also a check to ensure that the cursor otherwise does designate an element in the correct set (remember that cursors designate both an entity and the container); if this check fails then Program_Error is raised.

And now some new stuff, the usual set operations.

```
procedure Union(Target: in out Set;
                Source: in Set);
function Union(Left, Right: Set) return Set;
function "or" (Left, Right: Set) return Set
                 renames Union;

procedure Intersection(Target: in out Set;
                       Source: in Set);
function Intersection(Left, Right: Set) return Set;
function "and" (Left, Right: Set) return Set
                 renames Intersection;

procedure Difference(Target: in out Set;
                     Source: in Set);
function Difference(Left, Right: Set) return Set;
function "–" (Left, Right: Set) return Set
                 renames Difference;

procedure Symmetric_Difference(Target: in out Set;
                               Source: in Set);
function Symmetric_Difference (Left, Right: Set)
                                 return Set;
function "xor" (Left, Right: Set) return Set
                 renames Symmetric_Difference;
```

These all do exactly what one would expect using the equivalence relation on the elements.

```
function Overlap(Left, Right: Set) return Boolean;
function Is_Subset(Subset: Set; Of_Set: Set)
                                 return Boolean;
```

These are self-evident as well.

```
function First(Container: Set) return Cursor;
function Last(Container: Set) return Cursor;
function Next(Position: Cursor) return Cursor;
procedure Next(Position: in out Cursor);
function Find(Container: Set;
              Item: Element_Type) return Cursor;
function Contains(Container: Set;
                  Item: Element_Type) return Boolean;
```

These should be self-evident and are very similar to the corresponding operations on maps. Again unlike the operations on vectors and lists, Find logically searches the whole set and not just starting at some point (there is also no Reverse_Find). Moreover, Find uses the equivalence relation based on the "<" parameter.

```
function Has_Element(Position: Cursor) return Boolean;

procedure Iterate(Container: in Set;
                  Process: not null access procedure
                                  (Position: in Cursor));
```

These are also as for other containers.

The sets packages conclude with an internal generic package called Generic_Keys. This package enables some set operations to be performed in terms of keys where the key is a function of the element. Note carefully that in the case of a map, the element is defined in terms of the key whereas here the situation is reversed. An equivalence relationship is defined for these keys as well; this is defined by a generic parameter "<" for ordered sets and Equivalent_Keys for hashed sets.

In the case of ordered sets the formal parameters are

```
generic
  type Key_Type(<>) is private;
  with function Key(Element: Element_Type)
                                  return Key_Type;
  with function "<" (Left, Right: Key_Type)
                                  return Boolean is <>;
package Generic_Keys is
```

The following are then common to the package Generic_Keys for both hashed and ordered sets.

```
function Key(Position: Cursor) return Key_Type;
function Element(Container: Set; Key: Key_Type)
                                  return Element_Type;

procedure Replace(Container: in out Set;
        Key: in Key_Type; New_Item: in Element_Type);
procedure Exclude(Container: in out Set;
                  Key: in Key_Type);
procedure Delete(Container: in out Set;
                  Key: in Key_Type);

function Find(Container: Set; Key: Key_Type)
                                  return Cursor;
function Contains(Container: Set; Key: Key_Type)
                                  return Boolean;

procedure Update_Element_Preserving_Key(
                  Container: in out Set; Position: in Cursor;
                  Process: not null access procedure
                        (Element: in out Element_Type));
```

and then finally

```
end Generic_Keys;

private
  ... -- not specified by the language
end Ada.Containers.Ordered_Sets;
```

It is expected that most user of sets will use them in a straightforward manner and that the operations specific to sets such as Union and Intersection will be dominant.

However, sets can be used as sort of economy class maps by using the inner package Generic_Keys. Although this is certainly not for the novice we will illustrate how this might be done by reconsidering the stock problem using sets rather than maps. We declare

```
type Part_Type is
  record
    Part_Number: Integer;
    Year: Integer;
    Shelf: Character range 'A' .. 'T';
    Stock: Integer;
  end record;
```

Here we have put all the information in the one type.

We then declare "<" much as before

```
function "<" (Left, Right: Part_Type) return Boolean is
begin
  return Left.Part_Number < Right.Part_Number;
end "<";
```

and then instantiate the package thus

```
package Store_Sets is
      new Ordered_Sets(Element_Type => Part_Type);

The_Store: Store_Sets.Set;
```

We have used the default generic parameter mechanism for "<" this time by way of illustration.

In this case we add items to the store by calling

```
The_Store.Insert((34618, 1998, 'F', 25));
The_Store.Insert((27134, 2004, 'C', 45));
...
```

The procedure for checking the stock could now become

```
procedure Request(Part: in Integer: OK: out Boolean;
                  Year: out Integer; Shelf: out Character) is
  C: Cursor;
  E: Part_Type;
begin
  C := The_Store.Find((Part, others => <>));
  if C = No_Element then
    OK := False; return;                    -- no such item
  end if;
  E := Element(C);
```

```
    Year := E.Year;
    Shelf := E.Shelf;
    if E.Stock = 0 then
      OK := False; return;                    -- out of stock
    end if;
    Replace_Element(C, (E.Part_Number, Year;
                                    Shelf, E.Stock–1));
    OK := True;
  end Request;
```

This works but is somewhat unsatisfactory. For one thing we have had to make up dummy components in the call of Find (using <>) and moreover we have had to replace the whole of the element although we only wanted to update the Stock component. Moreover, we cannot use Update_Element because it is not defined for sets at all. Remember that this is because it might make things out of order; that wouldn't be a problem in this case because we don't want to change the part number and our ordering is just by the part number.

A better approach is to use the part number as a key. We define

```
type Part_Key is new Integer;

function Part_No(P: Part_Type) return Part_Key is
begin
  return Part_Key(P.Part_Number);
end Part_No;
```

and then

```
package Party is
      new Generic_Keys(Key_Type => Part_Key,
                       Key => Part_No);
use Party;
```

Note that we do not have to define "<" on the type Part_Key at all because it already exists since Part_Key is an integer type. And the instantiation uses it by default.

And now we can rewrite the Request procedure as follows

```
procedure Request(Part: in Part_Key; OK: out Boolean;
                  Year: out Integer; Shelf: out Character) is
  C: Cursor;
  E: Part_Type;
begin
  C := Find(The_Store, Part);
  if C = No_Element then
    OK := False; return;                      -- no such item
  end if;
  E := Element(C);
  Year := E.Year;  Shelf := E.Shelf;
  if E.Stock = 0 then
    OK := False; return;                      -- out of stock
  end if;

  -- we are now going to update the stock level
  declare
    procedure Do_It(E: in out Part_Type) is
    begin
      E.Stock := E.Stock – 1;
    end Do_It;
  begin
    Update_Element_Preserving_Key(The_Store, C,
                                  Do_It'Access);
  end;
  OK := True;
end Request;
```

This seems hard work but has a number of advantages. The first is that the call of Find is more natural and only involves the part number (the key) – note that this is a call of the function Find in the instantiation of Generic_Keys and takes just the part number. And the other is that the update only involves the component being changed. We mentioned earlier that there was no Update_Element for sets because of the danger of creating a value that was in the wrong place. In the case of the richly named Update_Element_Preserving_Key it also checks to ensure that the element is indeed still in the correct place (by checking that the key is still the same); if it isn't it removes the element and raises Program_Error.

But the user is warned to take care when using the package Generic_Keys. It is absolutely vital that the relational operation and the function (Part_No) used to instantiate Generic_Keys are compatible with the ordering used to instantiate the parent package Containers.Ordered_Sets itself. If this is not the case then the sky might fall in.

Incidentally, the procedure for checking the stock which previously used the maps package now becomes

```
procedure Check_Stock(Low: in Integer) is

  procedure Check_It(C: in Cursor) is
  begin
    if Element(C).Stock < Low then
      -- print a message perhaps
      Put("Low stock of part ");
      Put_Line(Element(C).Part_Number);      -- changed
    end if;
  end Check_It;

begin
  The_Store.Iterate(Check_It'Access);
end Check_Stock;
```

The only change is that the call of Key in

```
Put_Line(Key(C).Part_Number);
```

when using the maps package has been replaced by Element. A minor point is that we could avoid calling Element twice by declaring a constant E in Check_It thus

```
E: constant Part_Type := Element(C);
```

and then writing E.Stock < Low and calling Put_Line with E.Part_Number.

A more important point is that if we have instantiated the Generic_Keys inner package as illustrated above then we can leave Check_It unchanged to call Key. But it is important to realise that we are then calling the function Key internal to the instantiation of Generic_Keys (flippantly called Party) and not that from the instantiation of the parent ordered sets package (Store_Sets) because

that has no such function. This illustrates the close affinity between the sets and maps packages.

And finally there is a hashed sets package which has strong similarities to both the ordered sets package and the hashed maps package. We can introduce this much as for hashed maps by giving the differences between the two sets packages, the extra facilities in each and the impact on the part number example.

The specification of the hashed sets package starts

```
generic
  type Element_Type is private;
  with function Hash(Element: Element_Type)
                                    return Hash_Type;
  with function Equivalent_Elements(Left, Right:
                        Element_Type) return Boolean;
  with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Hashed_Sets is
  pragma Preelaborate(Hashed_Sets);
```

The differences from the ordered sets package are that there is an extra generic parameter Hash and the ordering parameter "<" has been replaced by the function Equivalent_Elements.

So if we have

```
function Equivalent_Parts(Left, Right: Part_Type)
                                    return Boolean is
begin
  return Left.Part_Number = Right.Part_Number;
end Equivalent_Parts;

function Part_Hash(P: Part_Type) return Hash_Type is
  M31: constant := 2**31–1;     -- a nice Mersenne prime
begin
  return Hash_Type(P.Part_Number) * M31;
end Part_Hash;
```

(which are very similar to the hashed map example – the only changes are to the parameter type name) then we can instantiate the hashed sets package as follows

```
package Store_Sets is
      new Hashed_Sets(Element_Type => Part_Type,
                      Hash => Part_Hash,
                Equivalent_Elements => Equivalent_Parts);

The_Store: Store_Sets.Set;
```

and then the rest of our example will be exactly as before. It is thus easy to convert from an ordered set to a hashed set and vice versa provided of course that we only use the facilities common to both.

It should also be mentioned that the inner package Generic_Keys for hashed sets has the following formal parameters

```
generic
  type Key_Type(<>) is private;
  with function Key(Element: Element_Type)
                                    return Key_Type
  with function Hash(Key: Key_Type)
                                    return Hash_Type;
  with function Equivalent_Keys(Left, Right: Key_Type)
                                    return Boolean;
package Generic_Keys is
```

The differences from that for ordered sets are the addition of the function Hash and the replacement of the comparison operator "<" by Equivalent_Keys.

(Incidentally the package Generic_Keys for ordered sets also exports a function Equivalent_Keys for uniformity with the hashed sets package.)

Although our example itself is unchanged we do have to change the instantiation of Generic_Keys thus

```
type Part_Key is new Integer;

function Part_No(P: Part_Type) return Part_Key is
begin
  return Part_Key(P.Part_Number);
end Part_No;

function Part_Hash(P: Part_Key) return Hash_Type is
  M31: constant := 2**31–1;     -- a nice Mersenne prime
begin
  return Hash_Type(P) * M31;
end Part_Hash;

function Equivalent_Parts(Left: Right: Part_Key)
                                    return Boolean is
begin
  return Left = Right;
end Equivalent_Parts;
```

and then

```
package Party is
      new Generic_Key(Key_Type => Part_Key,
                      Key => Part_No;
                      Hash => Part_Hash
                Equivalent_Keys => Equivalent_Parts);
use Party;
```

The hash function is similar to that used with hashed maps. The type Part_Key and function Part_No are the same as for ordered sets. We don't really need to declare the function Equivalent_Parts since we could use "=" as the actual parameter for Equivalent_Keys.

We will finish this discussion of sets by briefly considering the additional facilities in the two sets packages (and their inner generic keys packages) just as we did for the two maps packages (the discussion is almost identical).

The ordered sets package has the following additional subprograms

```
procedure Delete_First(Container: in out Set);
procedure Delete_Last(Container: in out Set);
function First_Element(Container: Set)
                                    return Element_Type;
function Last_Element(Container: Set)
                                    return Element_Type;
function Previous(Position: Cursor) return Cursor;
```

```
procedure Previous(Position: in out Cursor);
function Floor(Container: Set;
                    Item: Element_Type) return Cursor;
function Ceiling(Container: Set;
                    Item: Element_Type) return Cursor;
function "<" (Left, Right: Cursor) return Boolean;
function ">" (Left, Right: Cursor) return Boolean;
function "<" (Left: Cursor; Right: Element_Type)
                                          return Boolean;
function ">" (Left: Cursor; Right: Element_Type)
                                          return Boolean;
function "<" (Left: Element_Type; Right: Cursor)
                                          return Boolean;
function ">" (Left: Element_Type; Right: Cursor)
                                          return Boolean;
procedure Reverse_Iterate(Container: in Set;
              Process: not null access procedure
                                  (Position: in Cursor));
```

These are again largely self-evident. The functions Floor and Ceiling are similar to those for ordered maps – Floor searches for the last element which is not greater than Item and Ceiling searches for the first element which is not less than Item – they return No_Element if there is not one.

The functions "<" and ">" are very important for ordered sets. The first is equivalent to

```
function "<" (Left, Right: Cursor) return Boolean is
begin
  return Element(Left) < Element(Right);
end "<";
```

There is a general philosophy that the container packages should work efficiently even if the elements themselves are very large – perhaps even other containers. We should therefore avoid copying elements. (Passing them as parameters is of course no problem since they will be passed by reference if they are large structures.) So in this case the built-in comparison is valuable because it can avoid the copying which would occur if we wrote the function ourselves with the explicit internal calls of the function Element.

On the other hand, there is a general expectation that keys will be small and so there is no corresponding problem with copying keys. Thus such built-in functions are less important for maps than sets but they are provided for maps for uniformity.

The following are additional in the package Generic_Keys for ordered sets

```
function Equivalent_Keys(Left, Right: Key_Type)
                                          return Boolean;
```

This corresponds to the formal generic parameter of the same name in the package Generic_Keys for hashed sets as mentioned earlier.

```
function Floor(Container: Set;
                    Key: Key_Type) return Cursor;
function Ceiling(Container: Set;
                    Key: Key_Type) return Cursor;
```

These are much as the corresponding functions in the parent package except that they use the formal parameter "<" of Generic_Keys for the search.

Hashed sets, like hashed maps also have the facility to specify a capacity as for the vectors package. Thus we have

```
procedure Reserve_Capacity(Container: in out Set;
                    Capacity: in Count_Type);

function Capacity(Container: Set) return Count_Type;
```

The behaviour is much as for vectors and hashed maps. We don't have to set the capacity ourselves since it will be automatically extended as necessary but it might significantly improve performance to do so. Note again that Length(S) cannot exceed Capacity(S) but might be much less.

The other additional subprograms for hashed sets are

```
function Equivalent_Elements(Left, Right: Cursor)
                                          return Boolean;
function Equivalent_Elements(Left: Cursor;
                    Right: Element_Type) return Boolean;
function Equivalent_Elements(Left: Element_Type;
                    Right: Cursor) return Boolean;
```

Again, these are very important for sets. The first is equivalent to

```
function Equivalent_Elements(Left, Right: Cursor)
                                          return Boolean is
begin
  return Equivalent_Elements(Element(Left),
                                    Element(Right));
end Equivalent_Elements;
```

and once more we see that the built-in functions can avoid the copying of the type Element that would occur if we wrote the functions ourselves.

5 Indefinite containers

There are versions of the six container packages we have just been discussing for indefinite types.

As mentioned in Section 1, an indefinite (sub)type is one for which we cannot declare an object without giving a constraint (either explicitly or though an initial value). Moreover we cannot have an array of an indefinite subtype. The type String is a good example. Thus we cannot declare an array of the type String because the components might not all be the same size and indexing would be a pain. Class wide types are also indefinite.

The specification of the indefinite container for lists starts

```
generic
  type Element_Type(<>) is private;
  with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Indefinite_Doubly_Linked_Lists is
  pragma Preelaborate(Indefinite_Doubly_Linked_Lists);
```

where we see that the formal type Element_Type has unknown discriminants and so permits the actual type to be

any indefinite type (and indeed a definite type as well). So if we want to manipulate lists of strings where the individual strings can be of any length then we declare

```
package String_Lists is new
  Ada.Containers.Indefinite_Doubly_Linked_Lists(String);
```

In the case of ordered maps we have

```
generic
  type Key_Type(<>) is private;
  type Element_Type(<>)  is private;
  with function "<" (Left, Right: Key_Type)
                                  return Boolean is <>;
  with function "=" (Left, Right: Element_Type)
                                  return Boolean is <>;
package Ada.Containers.Indefinite_Ordered_Maps is
  pragma Preelaborate(Indefinite_Ordered_Maps);
```

showing that both Element_Type and Key_Type can be indefinite.

There are two other differences from the definite versions which should be noted.

One is that the Insert procedures for Vectors, Lists and Maps which insert an element with its default value are omitted (because there is no way to create a default initialized object of an indefinite type anyway).

The other is that the parameter Element of the access procedure Process of Update_Element (or the garrulous Update_Element_Preserving_Key in the case of sets) can be constrained even if the type Element_Type is unconstrained.

As an example of the use of an indefinite container consider the problem of creating an index. For each word in a text file we need a list of its occurrences. The individual words can be represented as just objects of the type String. It is perhaps convenient to consider strings to be the same irrespective of the case of characters and so we define

```
function Same_Strings(S, T: String) return Boolean is
begin
  return To_Lower(S) = To_Lower(T);
end Same_Strings;
```

where the function To_Lower is from the package Ada.Characters.Handling.

We can suppose that the positions of the words are described by a type Place thus

```
type Place is
  record
    Page: Text_IO.Positive_Count;
    Line: Text_IO.Positive_Count;
    Col: Text_IO.Positive_Count;
  end record;
```

The index is essentially a map from the type String to a list of values of type Place. We first create a definite list container for handling the lists thus

```
package Places is new Doubly_Linked_Lists(Place);
```

We then create an indefinite map container from the type String to the type List thus

```
package Indexes is new Indefinite_Hashed_Maps(
                  Key_Type => String;
                  Element_Type => Places.List;
                  Hash => Ada.Strings.Hash;
                  Equivalent_Keys => Same_Strings;
                  "=" => Places."=");
```

The index is then declared by writing

```
The_Index: Indexes.Map;
```

Note that this example illustrates the use of nested containers since the elements in the map are themselves containers (lists).

It might be helpful for the index to contain information saying which file it refers to. We can extend the type Map thus (remember that container types are tagged)

```
type Text_Map is new Indexes.Map with
  record
    File_Ref: Text_IO.File_Access;
  end record;
```

and now we can more usefully declare

```
My_Index: Text_Map :=
              (Indexes.Empty_Map with My_File'Access);
```

We can now declare various subprograms to manipulate our map. For example to add a new item we have first to see whether the word is already in the index – if it is not then we add the new word to the map and set its list to a single element whereas if it is already in the index then we add the new place entry to the corresponding list. Thus

```
procedure Add_Entry(Index: in out Text_Map;
                            Word: String; P: Place) is
  M_Cursor: Indexes.Cursor;
  A_List: Places.List;    -- empty list of places
begin
  M_Cursor := Index.Find(Word);
  if M_Cursor = Indexes.No_Element then
    -- it's a new word
    A_List.Append(P);
    Index.Insert(Word, A_List);
  else
    -- it's an old word
    A_List := Element(M_Cursor);          -- get old list
    A_List.Append(P);                     -- add to it
    Index.Replace_Element(M_Cursor, A_List);
  end if;
end Add_Entry;
```

A number of points should be observed. The type Text_Map being derived from Indexes.Map inherits all the map operations and so we can write Index.Find(Word) which uses the prefixed notation (or we can write Indexes.Find(Index, Word)). On the other hand auxiliary entities such as the type Cursor and the constant No_Element are of course in the package Indexes and have to be referred to as Indexes.Cursor and so on.

A big problem with the procedure as written however is that it uses Element and Replace_Element rather than Update_Element. This means that it copies the whole of the existing list, adds the new item to it, and then copies it back. Here is an alternative version

```
procedure Add_Entry(Index: in out Text_Map;
                    Word: String; P: Place) is
  M_Cursor: Indexes.Cursor;
  A_LIst: Places.List;     -- empty list of places
begin
  M_Cursor := Index.Find(Word);
  if M_Cursor = Indexes.No_Element then
    -- it's a new word
    A_LIst.Append(P);
    Index.Insert(Word, A_List);
  else
    -- it's an old word
    declare
      -- this procedure adds to the list in situ
      procedure Add_It(The_Key: in String;
                       The_List: in out Places.List) is
      begin
        The_List.Append(P);
      end Add_It;
    begin
      -- and here we call it via Update_Element
      Index.Update_Element(M_Cursor, Add_It'Access);
    end;
  end if;
end Add_Entry;
```

This is still somewhat untidy. In the case of a new word we might as well make the new map entry with an empty list and then update it thereby sharing the calls of Append. We get

```
procedure Add_Entry(Index: in out Text_Map;
                    Word: String; P: Place) is
  M_Cursor: Indexes.Cursor := Index.Find(Word);
  OK: Boolean;
begin
  if M_Cursor = Indexes.No_Element then
    -- it's a new word
    Index.Insert(Word, Places.Empty_List, M_Cursor, OK);
    -- M_Cursor now refers to new position
    -- and OK will be True
  end if;
  declare
    -- this procedure adds to the list in situ
    procedure Add_It(The_Key: in String;
                     The_List: in out Places.List) is
    begin
      The_List.Append(P);
    end Add_It;
  begin
    -- and here we call it via Update_Element
    Index.Update_Element(M_Cursor, Add_It'Access);
  end;
end Add_Entry;
```

It will be recalled that there are various versions of Insert. We have used that which has two out parameters being the position where the node was inserted and a Boolean parameter indicating whether a new node was inserted or not. In this case we know that it will be inserted and so the final parameter is a nuisance (but sadly we cannot default out parameters). Note also that we need not give the parameter Places.Empty_List because another version of Insert will do that automatically since that is the default value of a list anyway.

Yet another approach is not to use Find but just call Insert. We can even use the defaulted version – if the word is present then the node is not changed and the position parameter indicates where it is, if the word is not present then a new node is made with an empty list and again the position parameter indicates where it is.

```
procedure Add_Entry(Index: in out Text_Map;
                    Word: String; P: Place) is
  M_Cursor: Indexes.Cursor;
  Inserted: Boolean;
begin
  Index.Insert(Word, M_Cursor, Inserted);
  -- M_Cursor now refers to position of node
  -- and Inserted indicates whether it was added
  declare
    -- this procedure adds to the list in situ
    procedure Add_It(The_Key: in String;
                     The_List: in out Places.List) is
    begin
      The_List.Append(P);
    end Add_It;
  begin
    -- and here we call it via Update_Element
    Index.Update_Element(M_Cursor, Add_It'Access);
  end;
end Add_Entry;
```

Curiously enough we do not need to use the value of Inserted. We leave the reader to decide which of the various approaches is best.

We can now do some queries on the index. For example we might want to know how many different four-lettered words there are in the text. We can either use Iterate or do it ourselves with Next as follows

```
function Four_Letters(Index: Text_Map) return Integer is
  Count: Integer := 0;
  C: Indexes.Cursor := Index.First;
begin
  loop
    if Key(C)'Length = 4 then
      Count := Count + 1;
    end if;
    Indexes.Next(C);
    exit when C = Indexes.No_Element;
  end loop;
  return Count;
end Four_Letters;
```

We might finally wish to know how many four-lettered words there are on a particular page. (This is just an exercise – it would clearly be simplest to search the original text!) We use Iterate this time both to scan the map for the words and then to scan each list for the page number

```
function Four_Letters_On_Page(Index: Text_Map;
      Page: Text_IO.Positive_Count) return Integer is
  Count: Integer := 0;

  procedure Do_It_Map(C: Indexes.Cursor) is

    procedure Do_It_List(C: Places.Cursor) is
    begin
      if Element(C).Page = Page then
        Count := Count + 1;
      end if;
    end Do_It_LIst;

    procedure Action(K: String; E: Places.List) is
    begin
      if K'Length = 4 then
        -- now scan list for instances of Page
        E.Iterate(Do_It_List'Access);
      end if;
    end Action;

  begin
    Indexes.Query_Element(C, Action'Access);
  end Do_It_Map;

begin
  Index.Iterate(Do_It_Map'Access);
  return Count;
end Four_Letters_On_Page;
```

We could of course have used First and Next to search the list. But in any event the important point is that by using Query_Element we do not have to copy the list in order to scan it.

6 Sorting

The final facilities in the container library are generic procedures for array sorting. There are two versions, one for unconstrained arrays and one for constrained arrays. Their specifications are

```
generic
  type Index_Type is (<>);
  type Element_Type is private;
  type Array_Type is
        array (Index_Type range <>) of Element_Type;
  with function "<" (Left, Right: Element_Type)
                                    return Boolean is <>;
procedure Ada.Containers.Generic_Array_Sort
                          (Container: in out Array_Type);
pragma Pure(Ada.Containers.Generic_Array_Sort);
```

and

```
generic
  type Index_Type is (<>);
  type Element_Type is private;
  type Array_Type is
                array (Index_Type) of Element_Type;
  with function "<" (Left, Right: Element_Type)
                                    return Boolean is <>;
procedure Ada.Containers.
                    Generic_Constrained_Array_Sort
                      (Container: in out Array_Type);
pragma Pure(Ada.Containers.
                   Generic_Constrained_Array_Sort);
```

These do the obvious thing. They sort the array Container into order as defined by the generic parameter "<". The emphasis is on speed.

7 Summary table

This paper concludes with an appendix showing at a glance the various facilities in the six main containers.

References

[1] D. E. Knuth (1973). *The Art of Computer Programming, vol 3 – Searching and Sorting*, Addison-Wesley.

Appendix Container summary

In order to save space the following abbreviations are used in the table:

T	container type eg Map	H_T	Hash_Type
C: T	Container: container type	I_T	Index_Type
P: C	Position: Cursor	K_T	Key_Type
L, R	Left, Right	Ex_Index	Extended_Index
C_T	Count_Type	B	Boolean
E_T	Element_Type		

also Index – means that another subprogram exists with similar parameters except that the first parameters are of type Vector and Index_Type (or Extended_Index) rather than those involving cursors.

also Key and also Element similarly apply to maps and sets respectively.

	vectors	lists	hashed maps	ordered maps	hashed sets	ordered sets
generic	Y	Y	Y	Y	Y	Y
type Index_Type is range <>;	Y					
type Key_Type is private;			Y	Y		
type Element_Type is private;	Y	Y	Y	Y	Y	Y
with function Hash(...) return Hash_Type;			on Key		on Element	
with function Equivalent_...(L, R: ...) return Boolean;			on Key		on Element	
with function "<" (L, R: ...) return Boolean is <>;				on Key		on Element
with function "=" (L, R: E_T) return B is <>;	Y	Y	Y	Y	Y	Y
package Ada.Containers.... is	Vectors	Doubly_ Linked_ Lists	Hashed_ Maps	Ordered_ Maps	Hashed_ Sets	Ordered_ Sets
pragma Preelaborate(...);	Y	Y	Y	Y	Y	Y
function Equivalent_...(L, R: ...) return Boolean;				on Key		on Element
subtype Extended_Index ... No_Index: constant Ex_Ind := Ex_Ind'First;	Y					
type T is tagged private; pragma Preelaborable_Initialization(T);	Vector	List	Map	Map	Set	Set
type Cursor is private; pragma Preelaborable_Initialization(Cursor);	Y	Y	Y	Y	Y	Y
Empty_T: constant T;	Vector	List	Map	Map	Set	Set
No_Element: constant Cursor;	Y	Y	Y	Y	Y	Y
function "=" (Left, Right: T) return Boolean;	Y	Y	Y	Y	Y	Y
function Equivalent_Sets(L, R: Set) return Boolean; function To_Set(New_Item: E_T) return Set;					Y	Y

function To_Vector(Length: C_T) return Vector;
function To_Vector(New_Item: E_T;
Length: C_T) return Vector;
	Y					

function "&" (L, R: Vector) return Vector; function "&" (L: Vector; R: E_T) return Vector; function "&" (L: E_T; R: Vector) return Vector; function "&" (L, R: E_T) return Vector;	Y					
function Capacity(C: T) return C_T; procedure Reserve_Capacity(C: T; Capacity: C_T);	Y		Y		Y	
function Length(C: T) return Count_Type;	Y	Y	Y	Y	Y	Y
procedure Set_Length(C: in out T; Length: in C_T);	Y					
function Is_Empty(C: T) return B; procedure Clear(C: in out T);	Y	Y	Y	Y	Y	Y
function To_Cursor(C: Vector; Index: Ex_Ind) return Cursor; function To_Index(P: C) return Ex_Ind;	Y					
function Key(P: C) return K_T;			Y	Y		
function Element(P: C) return E_T;	Y also Index	Y	Y	Y	Y	Y
procedure Replace_Element(C: in out T; P: C; New_Item: E_T);	Y also Index	Y	Y	Y	Y	Y
procedure Query_Element(P: C; Process: not null acc proc(...));	in Element also Index	in Element	in Key, in Element	in Key, in Element	in Element	in Element
procedure Update_Element(C: in out T; P: C; Process: not null acc proc(...));	in out Elem also Index	in out Elem	in Key, in out Elem	in Key, in out Elem		
procedure Move(Target, Source: in out T);	Y	Y	Y	Y	Y	Y

	vectors	lists	hashed maps	ordered maps	hashed sets	ordered sets

procedure Insert(C: in out Vector; Before: Ex_Ind; New_Item: Vector);
procedure Insert(C: in out Vector; Before: Cursor; New_Item: Vector);
procedure Insert(C: in out Vector; Before: Cursor; New_Item: Vector; Position: out Cursor);
	Y					

procedure Insert(C: in out T; Before: C; New_Item: E_T; Count: C_T := 1);	Y also Index	Y				
procedure Insert(C: in out T; Before: C; New_Item: E_T; Position: out Cursor; Count: C_T := 1);	Y	Y				
procedure Insert(C: in out T; Before: C; Position: out Cursor; Count: C_T := 1); element has default value	Y also Index	Y				
procedure Insert(C: in out T; Key: K_T; New_Item: E_T; Position: out Cursor; Inserted: out B);			Y	Y	Y (no key)	Y (no key)
procedure Insert(C: in out T; Key: K_T; Position: out Cursor; Inserted: out B); element has default value			Y	Y		
procedure Insert(C: in out T; Key: K_T; New_Item: E_T);			Y	Y	Y (no key)	Y (no key)
procedure Prepend(C: in out Vector; New_Item: Vector);	Y					
procedure Prepend(C: in out T; New_Item: E_T; Count: C_T := 1);	Y	Y				
procedure Append(C: in out Vector; New_Item: Vector);	Y					
procedure Append(C: in out T; New_Item: E_T; Count: C_T := 1);	Y	Y				
procedure Insert_Space(C: in out V; Before: Cursor; Position: out Cursor; Count: C_T := 1);	Y also Index					
procedure Include(C: in out T; Key: Key_Type; New_Item: E_T);			Y	Y	Y (no key)	Y (no key)
procedure Replace(C: in out T; Key: Key_Type; New_Item: E_T);			Y	Y	Y (no key)	Y (no key)
procedure Exclude(C: in out T; Key: Key_Type);			Y	Y	Y (Item not key)	Y (item not key)
procedure Delete(C: in out T; P: in out C; Count: C_T := 1);	Y also Index	Y	Y (no count) also Key	Y (no count) also Key	Y (no count) also Element	Y (no count) also Element
procedure Delete_First(C: in out T; Count: C_T := 1); procedure Delete_Last(C: in out T; Count: C_T := 1);	Y	Y		Y (no count)		Y (no count)
procedure Reverse_Elements(C: in out T);	Y	Y				
procedure Swap(C: in out T; I, J: Cursor);	Y also Index	Y				
procedure Swap_Links(C: in out List; I, J: Cursor);		Y				

procedure Splice(Target: in out List; Before: Cursor; Source: in out List);
procedure Splice(Target: in out List; Before: Cursor; Source: in out List; Position: in out Cursor);
procedure Splice(Container: in out List; Before: Cursor; Position: in out Cursor);
		Y				

procedure Union(Target: in out Set; Source: Set);
function Union(L, R: Set) return Set;
function "or" (L, R: Set) return Set renames Union;
					Y	Y

	vectors	lists	hashed maps	ordered maps	hashed sets	ordered sets

procedure Intersection(Target: in out Set; Source: Set);
function Intersection(L, R: Set) return Set;
function "and" (L, R: Set) return Set renames Intersection;
					Y	Y

procedure Difference(Target: in out Set; Source: Set);
function Difference(L, R: Set) return Set;
function "–" (L, R: Set) return Set renames Difference;
					Y	Y

procedure Symmetric_Difference(Target: in out Set; Source: Set); function Symmetric_Difference (L, R: Set) return Set; function "xor" (L, R: Set) return Set renames Symmetric_Difference;					Y	Y
function Overlap(L, R: Set) return Boolean; function Is_Subset(Subset: Set; Of_Set: Set) return B;					Y	Y
function First_Index(C: T) return Index_Type;	Y					
function First(C: T) return Cursor;	Y	Y	Y	Y	Y	Y
function First_Element(C: T) return Element_Type;	Y	Y		Y		Y
function First_Key(C: T) return Key_Type;				Y		
function Last_Index(C: T) return Ex_Ind;	Y					
function Last(C: T) return Cursor;	Y	Y		Y		Y
function Last_Element(C: T) return Element_Type;	Y	Y		Y		Y
function Last_Key(C: T) return Key_Type;				Y		
function Next(P: C) return Cursor; procedure Next(P: in out C);	Y	Y	Y	Y	Y	Y
function Previous(P: C) return Cursor; procedure Previous(P: in out C);	Y	Y		Y		Y
function Find_Index(C: T; Item: E_T; Index: I_T := I_T'First) return Ex_Ind;	Y					
function Find(C: T; ... ; P: C := No_Element) return Cursor;	Element	Element	Key (no position)	Key (no position)	Element (no position)	Element (no position)
function Element(C: T; Key: K_T) return E_T;			Y	Y		
function Reverse_Find_Index(C: T; Item: E_T; Index: I_T := I_T'First) return Ex_Ind;	Y					
function Reverse_Find(C: T; ... ; P: C := No_Element) return Cursor;	Element	Element				
function Floor(C: T; ...) return Cursor; function Ceiling(C: T; ...) return Cursor;				Key: K_T		Item: E_T
function Contains(C: T; ...) return Boolean;	Element	Element	Key	Key	Element	Element
function Has_Element(P: C) return Boolean;	Y	Y	Y	Y	Y	Y
function Equivalent_... (L, R: Cursor) return Boolean; function Equivalent_... (L: Cursor; R:...) return Boolean; function Equivalent_... (L:...; R: Cursor) return Boolean;			Keys		Elements	
function "<" (L, R: Cursor) return Boolean; function ">" (L, R: Cursor) return Boolean; function "<" (L, Cursor; R: ...) return Boolean; function ">" (L, Cursor; R: ...) return Boolean; function "<" (L:...; R: Cursor) return Boolean; function ">" (L:...; R: Cursor) return Boolean;				Key		Element
procedure Iterate(C: in T; Process: not null acc proc (P: C));	Y	Y	Y	Y	Y	Y
procedure Reverse_Iterate(C: in T; Process: not null acc proc (P: C));	Y	Y		Y		Y

	vectors	lists	hashed maps	ordered maps	hashed sets	ordered sets

generic
  with function "<" (Left, Right: E_T) return B is <>;
package Generic_Sorting is
  function Is_Sorted(C: T) return Boolean;
  procedure Sort(C: in out T);
  procedure Merge(Target, Source: in out T);
end Generic_Sorting;
	Y	Y				

generic type Key_Type (<>) is private;					Y	Y
with function Key(Element: E_T) return Key_Type;					Y	Y
with function Hash(Key: K_T) return Hash_Type;					Y	
with function Equivalent_Keys (L, R: Key_Type) return Boolean;					Y	
with function "<" (L, R: Key_Type) return B is <>;						Y
package Generic_Keys is					Y	Y
function Equivalent_Keys(L, R: Key_Type) return B;						Y
function Key(P: C) return Key_Type;					Y	Y
function Element(C: T; Key: K_T) return Element_T;					Y	Y

procedure Replace(C: in out T; Key: Key_Type;
New_Item: E_T);
procedure Exclude(C: in out T; Key: Key_Type);
procedure Delete(C: in out T; Key: Key_Type);
					Y	Y

function Find(C: T; Key: K_T) return Cursor;					Y	Y
function Floor(C: T; Key: K_T) return Cursor; function Ceiling(C: T; Key: K_T) return Cursor;						Y
function Contains(C: T; Key: K_T) return Boolean;					Y	Y
procedure Update_Element_Preserving_Key (C: in out T; P: C; Process: not null acc proc (Element: in out E_T));					Y	Y
end Generic_Keys;					Y	Y
private ... -- *not specified by the language* **end** Ada.Containers....;	Y	Y	Y	Y	Y	Y

Living in towers – The story of multi project system builds

Per Sandberg

SaabSystems, SE-17588 Järfälla, Sweden; Tel: +46 8 5808 4648; email: per.sandberg@saabsystems.se

Rei Stråhle

SaabSystems, S:t Olofsg 9A, SE-75321 Uppsala, Sweden; Tel: +46 8 5808 7124; email: rei.strahle@saabsystems.se

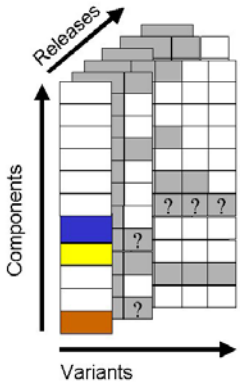
Abstract

Saab Systems has for a long period of time been delivering embedded software systems for military and civil usage. Starting 1985, an early migration to Ada was initiated, and we still regard Ada as the main language, though a mixed language environment with COTS components has become necessary.
This paper will demonstrate our present way of building the systems using an existing source library where active development for several customer projects is taking place. The library is comprised of more then 600 system units (components) with some variants. See also ref [1] and [2].
Keywords: embedded software, system build.

1 History

As mentioned earlier Saab Systems started in the mid 80:ies with Ada and at that point in time we where using Rational/R1000 as host machines. We immediately started to develop a methodology for building systems using Towers (though we did not know it was towers by then). In the beginning of the 90:ies we started to use APEX or PowerAda under AIX. This was a great step forward, with only 20 hours in build time compared to the earlier turn-around of 3 weeks.
In the mid 90:ies we started to use Intel-class PCs, both as host and target for new systems. We are still working with the same configuration, but the build tools of today is GNAT/GPS and the project facility fits like hand in a glove

- The building blocks in a Tower are components
- Components are released with new functionality or error corrections
- As a last resort, variants of components are created due to incompatible customer requirements



for our definition of Towers.

Figure 1 Components, releases and variants

The version control systems have been changing over time and been in the following sets: CMVC / SourceSafe / CVS / ClearCase. They are coexisting, and in some cases even within the same project.

2 Some Questions

- How can such a library be maintained, coherent and expanded?
- How can multiple development groups handle integration of parallel variants?
- How are the applications and the MMI components built together?
- How is the development of one system performed on different continents and time zones and different VC-systems?
- How come Ada05 is already used in the development?

3 Our Answers for Better or Worse?

The intent of the answers is to give a high level view on how we have solved the integration and maintenance problems in our multi-project environment.

3.1 How can such a library be maintained, coherent and expanded?

The key is that the interfaces between components are well defined and when changed - the changes must be backward compatible. If there are non-backward compatible changes in the interface, then all the changes must be carefully coordinated along with the projects and the components depending on the changed component/components needs to be upgraded within a short timeframe.

Applications and the MMI are built within a common framework that provides several data distribution mechanisms with different characteristics such as:

On high level:

- Reliability (with MMI interaction)
- Avoiding overload on the LAN with catch up if data is lost (Track Data)
- Time critical own position and speed.

On low level:

- Reliable singlecast datagrams
- Singlecast (almost as reliable...)
- Multicast

3.2 How can multiple development groups handle integration of parallel variants?

This is achieved by separating the version control system and the build system and by working with "Towers". A Tower is local "sandbox" where selected versions of the components are put. The selection of individual versions is done with a meta tools aware of the release conventions on component level. The selection list in text form may look like this:

```
comp_one.ss rev-1.2
comp_two.ss rev-3.2
comp_last.ss rev-6.2
```

The list looks a bit like the config.spec file in ClearCase but is restricted to contain subsystems only. There is a local working (sandbox) area where the developer/integrator stores the selected set of releases. This is very similar to the local working area where files are put from a VC system. In a Tower however, we are working with component-versions instead of file-version in order to get a maintainable number of versioned objects, or in practice versioned directories. The versioned directories (subsystems) are the smallest code-objects that are handled from a CM-point of view. A subsystem usually contains typically between 20 and 500 Ada units.
This is a picture of several towers that a developer may have in his/hers local work area. In reality the number of components is somewhere in between 100 and 300. The selection of the "active" tower is done using environment-variables.

	Tower 1	Tower 2	Tower 3
comp_one.ss	rev 1.2	rev 1.2	rev 1.2
comp_two.ss	rev 5.2	rev 1.5	rev 5.3
comp_three.ss	rev 1.2	rev 3.2	rev 3.2
comp_four.ss	rev 6.1	rev 6.0	rev 6.1
aasa_code.ss		rev 2.1	

The integration team for each customer project is responsible for the base configurations. However, a developer may chose to use different versions of some components to do pre-integration before the final releases to the customer projects.

3.3 How are the applications and the MMI components built together?

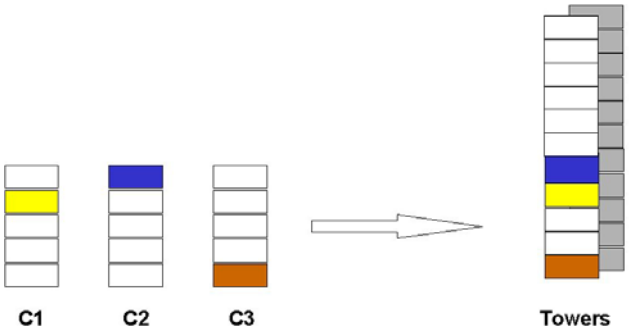
MMI and applications are connected with a formal interfacing description file.
The MMI is built using several languages Ada / C++ / Own languages. The sources from the different MMI-development groups are fetched into their proper subsystems in the tower and then compiled using the appropriate compiler.

3.4 How is the development of one system performed on different continents and time zones?

Development on different sites using different VC-systems requires a common transport format that must agree, and with a release naming that is consistent between the sites. Another key issue is that it is only possible to make changes in one component at one site at any given point in time; this has to be managed manually since different VC-systems are used. (Not optimal, but just a minor issue.)

3.5 How come Ada05 is already used in the development?

We owe this to the fact that the GNATPro "wavefront" is gradually including Ada05-features a bit ahead of schedule. This has so far not been a drawback, but merely a way of introducing the new features for the developers. The risk of misuse of the new possibilities is not really seen as a real project risk, but instead much appreciated by the developers. Progress means looking ahead!



- Error corrections can not be made directly
- Releases must be synchronised
- Working components may be changed

Figure 2 Components and towers

Summary

Keep the interfaces well defined!
Keep the number of items per configuration view limited! (By experience the maximum number is around 400 items.)

References

[1] Bass, Len; Clements, Paul; Kazman, Rick: "Software Architecture in Practice", ISBN 0-201-19930-0, Addison-Wesley, 1998.

[2] Källberg, Björn, Stråhle, Rei: "*Ship System 2000, a Stable Architecture under Continuous Evolution*", Ada-Europe 2001, ISBN 3-540-42123-8, Springer Verlag.

On the benefits for industrials of sponsoring free software development

J-P. Rosen

Adalog, 19-21 rue du 8 mai 1945, 94110 ARCUEIL, France; Tel: +33 1 41 24 31 40; email: rosen@adalog.fr

Abstract

Adalog [1] has developed two tools recently, one for an industrial client[1] (AdaSubst/AdaDep), and one for Eurocontrol (AdaControl). Although the programs were custom-made after the requirements of the clients, in both cases, they allowed the tools to be released as free software after they were delivered to them. In this presentation, we describe the clients' needs, the tools that were produced, and more importantly our experience that releasing the tools as free software was indeed beneficial to the clients, to Adalog, and to the community at large.

Keywords: free-software, industrial experience, semantic tools, ASIS.

Introduction

When an industrial company develops a software tool, it usually keeps it for itself. The rationale is simple: if the company pays for the software, it owns the software. Why would a company pay for the benefits of others, by making it freely available?

First it should be noted that, contrary to a common misunderstanding, releasing a tool as free software does *not* mean that the company does not own it anymore: free software is not public domain software. The company holds the copyright, and can do whatever it wants with it, including reusing it in part or in whole for proprietary programs. Making software free *never* diminishes the rights of the owner, including the right of not making new releases free (unlike *users* of the software who must continue to distribute it freely, at least when the software is provided under the terms of the GPL).

However, releasing the tool freely outside the company implies that anybody can use it, including the company's competitors; this may create concerns. On the other hand, this also means that anybody can contribute to it and improve it. Therefore, taking the decision of releasing a program as free software is really a matter of balancing benefits and drawbacks.

In this paper, we describe two experiments where the releasing of paid developments as free software was beneficial to the industrials. We do not claim that this can be done in every case, but we argue that "paying for free software" can be cost effective for certain classes of tools.

[1] Who didn't want to be disclosed

1 The case of AdaSubst/AdaDep

1.1 Context

An industrial client had developed over the years several big libraries dealing with its problem domains. Since this effort started long ago, the code and the structure of the libraries were still compatible with Ada83. And, as is often the case when a code has evolved over many years, it came to a point where a major restructuring was needed.

Axlog [2], Adalog's mother company, won the contract for reorganizing this software components base. This implied, among other things, breaking big packages into a hierarchy of child packages, and often changing the names of the provided services. Of course, such changes would break all existing code that used the libraries. Therefore, the contract stipulated that a tool should be provided to migrate code to the new library structure. The initial intent was to provide some kind of ad-hoc Python program to do this.

1.2 Solution

Adalog proposed to develop instead a general tool (AdaSubst), based on ASIS (*Ada Semantic Interface Specification* [3]), which would not be specific to this migration, but could be used for any similar needs. A dictionary file describes, for each entity, its old name and the place where it was declared, and its new mapping, i.e., its new name and the new package where it is now. Typical entries in the dictionary look like this:

```
Old_Pack => New_Pack
Old_Pack.Proc1 => New_Pack.Proc2
Pack1.Func{integer return integer} => New_Func
Big_Pack => Parent, Parent.Child1, Parent.Child2
all Print => Put
```

The first line means that the package "Old_Pack" is now called "New_Pack"; the second line means that the procedure "Proc1" in package "Old_Pack" has been renamed to "Proc2" in package "New_Pack". The third line is an example of dealing with overloaded declarations: only the function "Func" that takes an Integer parameter and returns an Integer value is changed into "New_Func". In the case of the fourth line, a package has been split into a parent package and two child units. The last line means that all procedures named "Print" are now called "Put", irrespectively of where they are located. Note that if a package name changes, it is not necessary to specify the transformation for all its elements, as long as the names are not changed; only changed elements need to be described. The tool makes all the necessary transformations on the code, taking all Ada rules into account; use clauses are

properly modified, overloading is taken into account; when a name changes in a generic, the change is propagated to all uses in all instances, etc. The only case that is not fully automated is for elements declared in a package that has been split (like "Big_Pack" above). "With" and "use" clauses are transformed to name all new packages, but for an element given in prefixed notation, it is not possible to know in which unit it resides know. In this case, the transformation prefixes the name by the various possible packages, separated by '?'. Since this does not compile, it is easy to edit the construct to choose the appropriate package manually. In short, the tool goes far beyond what could be done by text substitution, even with sophisticated pattern matching tools such as those provided by Python.

In addition, the migration itself required a detailed analysis of which elements from all "withed" packages were used. Adalog developed a companion tool (AdaDep) to ease this analysis. It gives, for a given unit, which elements from each withed unit is used and how many times, together with the nature of the element. For example, given:

```
package Pack is
  I : Integer;
  package Internal is
    V : Float;
  end Internal;
end Pack;

with Pack, Text_IO;
use Pack, Text_IO;
procedure Sample is
begin
  I := 1;
  Internal.V := 3.0;
  Put_Line (Integer'Image (I + Integer(Internal.V)));
end Sample;
```

Running AdaDep will produce:

```
SAMPLE (body) =>
=> from ADA.TEXT_IO
  PUT_LINE - A_Procedure_Declaration * 1
=> from PACK
  I - A_Variable_Declaration * 2
=> from PACK.INTERNAL
  V - A_Variable_Declaration * 2
=> from STANDARD
  INTEGER - An_Ordinary_Type_Declaration * 2
```

In agreement with the client, AdaSubst and AdaDep were released as free software. The client, who is not in the language tools business, had no interest in keeping them proprietary.

1.3 Lessons learned

In the end, the provided tool was far more powerful than initially required. Although the requirement was to simply minimize manual adjustments, it turned out that AdaSubst properly processed automatically several 100 000's SLOCs without *any* correction (except for ambiguities).

As for any other contract, the tool was delivered to the client with a warranty period. It happened that shortly after the end of this warranty period, the client reported a bug. Had the tool been developed under a conventional contract, we would have asked for a contract extension to make a fix. However, since at that time the tool was free software, we reacted like any developer of free software: we said "thank you for reporting this", and fixed the problem. This little story shows that by allowing the tool to be released as free software, the client eventually got better (and free) support than under a regular contract.

Even after the end of the contract, the tool continued to evolve and improve, thanks to the community feed-back. The client now has a better and more general tool than if it had kept it proprietary.

The approach was also beneficial to Adalog: the tools are commonly used inside the company, and many parts of them could be reused in other developments. For example, Adalog helped one of its clients in a migration to a different target; representation clauses from the original system were no more appropriate. It was easy to adapt Adasubst to provide a new functionality that commented out all representation clauses from the original program.

2 The case of AdaControl

2.1 Context

Eurocontrol (*European Organisation for the Safety of Air Navigation*) is developing programs to manage air traffic all over Europe. These programs are not life-critical, in the sense that a failure would not cause planes to crash, however a break-down of the system would cause huge delays for all airplanes flying over Europe; the software is therefore highly business critical. The system is made of very big programs (over 1.1 MSLOC), developed and maintained by a big team. With a project of this scale, it is not possible to rely on individual discipline to make sure that programming rules are being followed; Eurocontrol needed a tool to enforce programming rules and search for occurrences of bad or arguable programming practices. Thanks to the cooperation with AdaCore, some of these checks were incorporated into the GNAT compiler. However, many rules were deemed too specific to be put in a compiler, and it was felt that an independent controller program, allowing parameterizable rules, was necessary.

There can be many such rules, and it was expected that new ones would appear as more experience was gained by using the tool (and this expectation was verified quite rapidly). Therefore, the contract called for a general framework, where rules could be added at will with minimum effort, with just a minimum number of rules to be implemented as part of the original contract, to serve as a proof of concept.

The bid was granted to Adalog. It is interesting to note that since AdaSubst was free software, Adalog could show it in its response to the bid, as a show-case of its know-how in ASIS development.

Like the first client, Eurocontrol is not in the business of providing tools. On the other hand, such a tool was deemed

useful to the community at large. Moreover, since the tool is easily extendable, Eurocontrol felt that it would benefit from the contributions of other users. Therefore, it was decided right from the start that the program would eventually be released as free software.

2.2 Solution

Like AdaSubst, AdaControl is based on ASIS. Actually, it is a perfect example of the kind of application that ASIS was intended for.

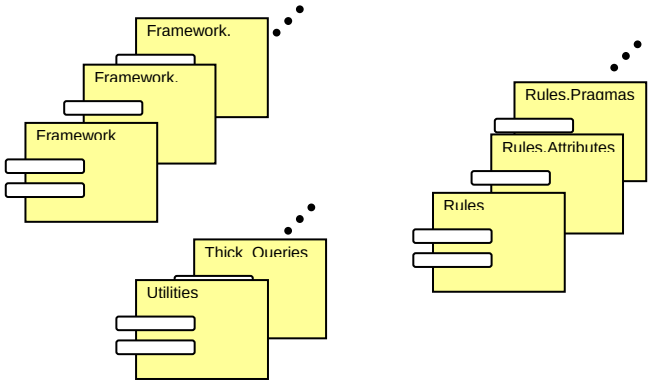
The structure of AdaControl has been designed to make the addition of new rules as simple as possible. It provides a general framework that hides all the internal machinery and offers a number of utilities that make the writing of rules easier: various services are provided to deal with complex issues like overloading, scope management, etc. Rules are plugged in a special module, and rule writers have to care only with the ASIS requests necessary to the rule.

Actually, AdaControl implements a full language to describe the checks that are to be performed. Utilities are provided to the rules for easy parsing of the rules' parameters. There is an interpreter for this language, allowing AdaControl to be used interactively as well as in batch mode. Rules just register themselves to the interpreter, thus adding new "verbs" to the command language, without needing to change the interpreter itself.

An important feature of AdaControl is that rules can be locally disabled by means of special comments in the code. This allows for local derogations to a rule, which is very important since there are almost always cases where general rules are not applied for good reasons. The mechanism for this is hidden in the module that reports errors, therefore the writer of a rule does not have to care about it: it is fully automatic.

Finally, the framework provides facilities for debugging rules. This is a great help since, given the complex structures used by ASIS, it is difficult to understand the origin of a problem under a debugger.

The overall structure of AdaControl is thus made of three well identified and separated parts: the framework itself (specific to AdaControl), general ASIS utilities (useful for any ASIS application), and the rules, as pictured below:



Important modules from AdaSubst were reused in AdaControl; this raised no copyright issue, since both programs were free software. In proprietary development, it is often the case that similar modules must be developed again, since it is not possible to provide a client with a module developed for another client!

2.3 Lessons learned

In addition to the framework, the initial bid required the implementation of only four rules. Later, an extension to the contract supported the development of three more rules. But since Adalog had similar needs for controlling its own programs, we developed other rules for our own benefit. The result was that the tool was delivered with more rules than contractually required, and the number of rules continued to grow after the end of the contract. At the time of writing (version 1.4), 25 rules are implemented (each with various parameters that allow them to check many things). It is expected that the number of rules will continue to grow as the tool gets more and more used.

As mentioned above, several modules were reused from Adasubst, especially those dealing with command line options and the way of specifying which units are to be processed (including integration with GNAT's ".adp" project files). On top of the usual benefits of reuse (no need to rewrite, test, debug), this brought two benefits that are rarely mentioned:

Uniformity. Since the modules are the same, the user instructions for using Adasubst and AdaControl are the same.

Reuse of documentation. Similarly, part of the user documentation for AdaControl was reused from the documentation from AdaSubst.

Thanks to the continued cooperation with Eurocontrol, all the rules were checked against Eurocontrol's software, thus providing an extensive test bench that would not have been available if Adalog had developed the product in-house.

2.4 The consortium effect

Since its initial release, the tool has raised interest in several other companies, which are willing to sponsor further development, including the development of more rules. At this point, the story of AdaControl seems to open the way to a new model of commercial free-software: cooperative development. The situation is that several companies, from totally different markets, needed a tool; none of them was willing to pay for the full development, and their interests were too different to even think of gathering them all in a consortium, just for the sake of developing the tool. This is however exactly what happened in practice: one company put the initial stake, other companies contribute in proportion of their particular needs, and in the end everybody benefits from a much more sophisticated tool than could have been developed (custom or in-house) by each of the companies separately.

3 Adalog's point of view

As explained above, releasing the tools as free software had a number of benefits for the client. But from a vendor's point of view, isn't it better to have a product that can be sold under a usual proprietary license?

First, it should be noted that developing a tool with the intent of selling it requires an important upfront investment. Such tools require many months of work, before even knowing if the tool will raise interest on the market place. By having the development paid under contract, Adalog could minimize the risk, and by having the tool released as free software, Adalog continued to have the opportunity of turning the development into a commercial product that can be offered to others than the initial customer.

Building a successful product for a client is always good for a company, but only the client knows about the quality of the work. If the product is released as free software, then anybody can assert the quality of the product. This makes good publicity for the company… and also attracts the sympathy of the community at large. It demonstrates Adalog's know-how in the development of custom language tools and its ability in ASIS development. The tools now form a suite of Ada semantic utilities, and we hope that they will attract new clients who need other similar tools (and will hopefully accept that they be released as free software too).

Since we consider these tools as fully commercial, Adalog is selling support contracts for them, and develops (paying) improvements on demand for clients with special needs. All this means more business opportunities.

There is also a "business attracts business" effect. Adalog has developed a custom analysis tool for a client, based on the same technology as AdaControl. The availability of AdaControl not only demonstrated the ability of Adalog for designing semantic tools, but also gave the client the idea of having a tool made to his own needs.

Finally, a side benefit is that the availability of these programs on Adalog's web site [4] attracts many people to visit us. Adalog uses a Web measurement service [5] to measure the popularity of its site; among 269 sites in the '"programming languages" category, Adalog ranks 9th, which is a good indication of its own popularity… as well as of the interest for Ada.

4 Difficulties

Sponsoring free software may create some difficulties, because it goes against a number of established practices. For example, all standard contracts stipulate that the product becomes the property of the client. This in itself does not preclude the software from being free, but in practice, for free software to grow and flourish, it is necessary to have a well identified, centralized entity to which contributions can be sent. To most users, this will be the name that appears in the copyright notice. It is therefore more convenient if the company that made the initial development keeps the copyright (possibly shared with the client, as was done with Eurcontrol). This must be negociated with the client.

The legal department of the client company may also on occasions be unaware of what free software really means, and raise concerns. It is then necessary to either educate the lawyers, or find an arrangement that does not raise issues of intellectual property. For example, it is possible to have a contract by which the provider must "provide a tool" (including a free one) to the client, and not state contractually that the tool is actually developed for the client. And of course, there is the issue of finding who, in the client's company, is empowered to sign the letter allowing the product to be released as free software…

Finally, it is clear that there are many tools whose nature is not appropriate for this model of development; this can work only for tools that are general enough to not require any problem domain knowledge (which clearly belongs to the client), and be usable in different application fields.

Conclusion

The story of AdaSubst/AdaDep and Adacontrol is another example that it is possible to develop *commercial*, but *free* software. Of course, Adalog is not the first company to take this approach: obvious other examples are RedHat (and others) with Linux distributions, MySQL for databases, and AdaCore with the GNAT compiler. However, our approach is different by using a business model that allows sponsoring the developments by various, unrelated companies, thus building a "virtual consortium".

In conclusion, releasing these tools as free software was beneficial to the industrials, because they have tools that are more powerful than if they had kept them proprietary. Moreover, they benefit from continued maintenance and improvements. But it is also beneficial to Adalog, as a showcase of what the company can achieve, and because it generates more business through custom improvements and maintenance contracts. And last but not least, it is beneficial to the Ada community at large, since anybody can use the tool.

Web references

[1] http://www.adalog.fr:
Adalog's home page

[2] http://www.axlog.fr:
Axlog's home page

[3] http://www.sigada.org/wg/asiswg/asiswg.html:
ASIS working group.

[4] http://www.adalog.fr/compo2.htm:
Access to Adalog components, including AdaSubst, AdaDep, and AdaControl.

[5] http://www.weborama.fr:
Web measurement service.

Ada-Europe 2005 Sponsors