Minutes of the 27th ARG Meeting

16-18 April 2005

Burlington, Massachusetts, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Gary Dismukes, Bob Duff, Kiyoshi Ishihata, Pascal Leroy, Steve Michell, Erhard Ploedereder, Tucker Taft.

Observers: None.

Meeting Summary

The meeting convened on 16 April 2005 at 09:15 hours and adjourned at 18:00 hours on 18 April 2005. The meeting was held in the conference room at SofCheck in Burlington, Massachusetts, USA. The meeting covered the entire agenda as well as various off-agenda items.

AI Summary

The following AIs were approved:

AI-417/01 Lower bound of functions in Ada.Tags and Ada.Exceptions (10-0-1)
AI-429/01 Representation of minus signs (11-0-0)

The following AIs were approved with editorial changes:

AI-381/04 New Restrictions identifier No_Dependence (10-1-0)
AI-395/06 Various clarifications regarding 16- and 32-bit characters (10-0-1)
AI-403/02 Preelaboration checks and formal objects (11-0-0)
AI-416/06 Access results, accessibility, and return statements (8-0-1)
AI-418/02 Vector norm (10-0-1)
AI-419/03 Limitedness of derived types (11-0-0)
AI-420/01 Equality of anonymous access types (8-0-3)
AI-421/03 Sequential activation and attachment (10-0-0)
AI-423/04 Renaming, null exclusion, and formal objects (9-0-1)
AI-424/01 List of language-defined units (11-0-0)
AI-425/01 Organization of Annex M (8-1-2)
AI-426/01 Language-defined routines returning abnormal and invalid values (9-0-2)
AI-427/02 Default parameters and Calendar operations (11-0-0)
AI-428/01 Input-output for bounded strings (10-0-1)
AI-430/00 Conventions of inherited subprograms (9-0-0)
AI-431/00 Conversions to remote access-to-subprogram types (9-0-1)
AI-432/00 Out of range values in Ada.Real_Time (10-0-0)

The corrections to the following approved AIs were approved with changes (as a group unless otherwise noted — 10-0-0):

AI-51-2/03 Size and Alignment clauses
AI-161/09 Default-initialized objects
AI-162/07 Anonymous access types and task termination/controlled type finalization
AI-195/15 Streams
AI-216/16 Unchecked unions -- variant records with no run-time discriminant
AI-230/16 Generalized use of anonymous access types
AI-237/08 Finalization of task attributes
AI-247/03 Alignment of composite types
AI-248/12 Directory operations
AI-251/19 Abstract Interfaces to provide multiple inheritance (9-0-2)

AI-266-2/10 Task termination procedure
AI-270/06 Stream size item control
AI-285/15 Support for 16-bit and 32-bit characters
AI-287/12 Limited aggregates allowed (10-0-0)
AI-291-2/04 By-reference types and the recommended level of support for representation items
AI-296/10 Vector and matrix operations
AI-297/12 Timing events
AI-301/12 Operations on language-defined string types
AI-302-3/12 Container library (6-0-4, 5-1-5)
AI-305/09 New pragma and additional restriction identifiers for real-time systems
AI-307/11 Execution-time clocks
AI-327/08 Dynamic ceiling priorities
AI-337/03 Applicability of C interfacing advice to private types
AI-344/11 Allow nested type extensions (10-0-1)
AI-345/08 Protected and task interfaces
AI-354/08 Group execution-time budgets
AI-355/08 Priority Specific Dispatching including Round Robin
AI-357/09 Support for Deadlines and Earliest Deadline First Scheduling
AI-360/07 Types that need finalization
AI-376/02 Interfaces.C works for C++ as well
AI-385/04 Stand-alone objects of anonymous access types
AI-386/03 Further functions returning Time_Span values
AI-394/03 Redundant Restriction Identifiers and completing Ravenscar definition
AI-397/03 Conformance and overriding for entries and protected operations (11-0-0)
AI-399/02 Single task and protected objects implementing interfaces
AI-400/03 Wide_ and Wide_Wide_ images of identifiers
AI-401/03 Terminology for interfaces
AI-404/04 Not null and all in access parameters and types
AI-407/04 Terminology and semantics of prefixed views
AI-409/02 Conformance and overload resolution related to anon access types
AI-412/04 Subtypes of incomplete types; renamings of limited views
AI-414/02 pragma No_Return for overriding procedures

The following AIs were voted No Action:

AI-302-4/00 Container library (mail container) (11-0-0)
AI-422/01 User data for timing events (11-0-0)

Detailed Minutes

Welcome

Tucker welcomes the ARG to SofCheck, and describes plans for a full slate of social events. He invites us to dinner at his house Saturday night, and has scheduled a historical bus tour in honor of the (local) holiday. (It is Patriots Day, which commemorates the start of the American Revolution.) Pascal reminds us that we came to Burlington to work.

John has asked Tucker to borrow him a laptop for the meeting. Tucker provided a full-size computer which took up a significant portion of the conference table. John gripes that it would have been unnecessary because Bob Duff showed up with a printout of the entire AARM. But Bob wasn't going to print out another copy for John to use.

Bob gripes that *his* version of the AARM (Ada 95) *only* took 800 pages. The draft 11 AARM takes 1050 pages. Randy points out that a significant portion of the new content is the new packages: Containers, Directories, Matrices, and Time Operations; and these don't represent many new concepts. Bob continues to grumble about the number of trees that is required to print the AARM. Many of us wonder why *any* trees are necessary.

Meeting Minutes

John notes that there is a period missing after the discussion of AI-235.

In the discussion of AI-395 "protect[]ted" is spelled wrong, the last 't' should be removed.

In item 4 of AI-395, “wwc_12345689” should be “wwc_12345678”.

Approve the minutes of the 26th meeting with changes: 11-0-0.

Schedule of the Amendment

Randy didn’t finish the Annex sections until April 4th. He and Pascal decided to wait with a release until just before the meeting, as there wasn’t enough time left for a full review anyway; Randy added the text from all of the Paris approved AIs, and most of the other AIs discussed in Paris, to the draft 11 AARM. The text from AI-395 and AI-416 was incompletely added; core only for AI-395, and all but 3.10.2, 4.8, and 6.5 for AI-416.

Pascal notes that since all of the AARM sections are now finished, all remaining reviews should be completed by May 6th. Tucker will complete a presentation AI covering additional core language examples by May6th as well.

The review of the remaining clauses should be done against draft 11.

Randy will try to have draft 12 by the May 15th, with at least the core finished (comments and newly approved AIs). Pascal will inform WG9 of the availability of this draft so that they can start their review early if they wish.

Randy asks how we plan to review the actual Amendment document, since it is generated from a separate source than the AARM. Bob suggests comparing the Amendment paragraphs to the RM paragraphs be automated if possible. We think that is a very good idea, and everyone looks at Bob. Bob realizes that he’s volunteered to do so, and he agrees to undertake the job.

Bob has previously volunteered to attempt (his words) to review the entire AARM. (That will be one loonnnggg evening.) He should do that on draft 12 for the core and draft 13 for the Annexes (or whatever the division turns out to be).

Randy should try to have a complete draft 13 by June 3rd. That would not necessarily include Bob’s review comments).

Of course, there will need to be a draft 14 just before the York ARG meeting (June 15th).

Informal Nomenclature of the language with the Amendment

Should the language resulting from adding the Amendment to Ada 95 before informally known as Ada 2006 or Ada 2005? Randy is worried that the name will change anyway; we should do it sooner rather than later. Pascal worries about confusion between Ada 2006, and Ada 2005. In the standard, the annotations say Ada 2005, and the heading will say 2006. John says that he thought we agreed in the past to say Ada 2005 no matter what. Bob says Ada 95 was handed to the ISO in Dec 1994, but it wasn’t published until 1995, and the name was changed to reflect that. That step wouldn’t happen this time until January 2006 at the earliest. Steve Michell says we should just be honest and say Ada 2006.

Straw vote: Ada 2006: 4; Ada 2005: 1; Abstain: 6.

Pascal says that this is a clear vote to change to using Ada 2006.

John asks that we check with Jim that this isn’t a problem. [Editor’s note: After the meeting, Jim says that he would prefer this topic to be handled by WG9, so a decision is effectively postponed until the York WG9 meeting.]

Next Meeting

The next scheduled meeting is in York, before the Ada Europe conference. It’s clear now that we’ll need this meeting. Because of local events, Alan suggests starting at noon on Saturday at York. He suggests staying in London on Friday night, taking a train to York in the morning.

We tentatively agree to this plan; we’ll start Saturday, June 18th at noon, working through Monday, June 20. We decide that we’ll reconsider the dates on Monday, based on the progress made during this meeting.

Most likely, the meeting after that would be at SIGAda; November 18-20 in Atlanta. Someone wonders if we'll have enough work. Pascal says that we might end up getting ASIS, as nothing seems to be happening in its working group.

Steve Michell volunteers Mont Tremblant in Quebec for a late September meeting if needed. (Sept. 25 - Oct. 2 would be available.) Hopefully, we won't need that.

On Monday afternoon, Pascal notes that we probably will not have a lot of technical issues to settle in York, rather just a long line of editorial corrections to discuss and approve. We agree to shorten the meeting to a 2-day meeting, starting on Sunday morning, June 19th.

Thanks

The ARG thanks our hosts, SofCheck and Tucker Taft for hosting the meeting and arranging the fine social events (including the cookout and the interesting historical tour).

The ARG gives especial thanks to Tullio Vardanega for deciding not to come at the last moment, as we had just barely enough room. If he had come, we would have had to meet in the parking lot.

Old Action Items

Steve Baird didn't do AI-383. Randy was very late getting the AARM posted with all of the sections completed. No one reviewed section 13, or Annexes B, C, or D because the AARM containing them wasn't ready in time. Steve Michell did not do the AI he was assigned. It will be dropped from the action items (and will not appear in the Amendment). Tucker did not review/create any examples. He pleads that they should be done last; that's OK, but last has now arrived. All other items were finished.

New Action Items

The combined unfinished old action items and new action items from the meeting are shown below.

Steve Baird:

- AI-383
- Review AARM Annex B.

John Barnes

- Re-review automatically generated annexes of AARM (K, L, P, index).

Randy Brukardt:

- Post draft 12 of the consolidated AARM including all AI text approved to date and editorial comments on the core by May 15th.
- Post draft 13 of the consolidated AARM including all editorial comments received by May 15th around June 3rd.

Editorial changes only:

- AI-195
- AI-248
- AI-251
- AI-270
- AI-287
- AI-291-2
- AI-305
- AI-327
- AI-381

- AI-395
- AI-397
- AI-403
- AI-416
- AI-419
- AI-420
- AI-421
- AI-423
- AI-424
- AI-425
- AI-426
- AI-427
- AI-428
- AI-430
- AI-431
- AI-432

Alan Burns:

- Review AARM Annex D.

Gary Dismukes:

- Review AARM Annex C

Bob Duff:

- Review entire core of the draft 12 AARM, and the annexes of the draft 13 AARM.
- Attempt to automate comparing the Amendment text to the RM text.

Pascal Leroy:

- Write a correction to AI-394 to get rid of "apply in this Annex" wording in H.4 (see Miscellaneous Topics).
- Review AARM Section 13.

Steve Michell:

- Review AARM Section 13.

Erhard Ploedereder:

- Review AARM Annex C and Annex D.
- Review wording changes of AI-401 once they are integrated into the AARM.

Tucker Taft:

- Review AARM Annex B.
- Review the examples in the entire Standard and Amendment; suggest new examples as needed (especially for new features of Ada 2006). Complete by May 6th.

Detailed Review

The minutes for the detailed review of AIs are divided into existing amendment AIs and non-amendment AIs. The AIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as "for"-"against"-"abstentions". For instance, a vote of 6-1-2 would have six votes for, one vote against, and two abstentions.

Corrections to Approved AIs

Unless otherwise noted, all corrections were approved as a group (including the changes to them noted below). The vote was 10-0-0.

AI-51-2/03 Size and Alignment clauses

Randy moved Implementation Advice about the size of arrays to 13.3 (with other Implementation Advice on size).

The new wording for 13.3(30) was redundant with 13.1(21.1); several paragraphs were changed.

AI-161/09 Default-initialized objects

The other bullets in C.4 start with a lower case letter, so the new one ought to as well.

AI-162/07 Anonymous access types and task termination/controlled type finalization

The Implementation Advice added to 13.11.2(17) was split into a separate paragraph, since it has nothing to do with the standard storage pool.

AI-195/15 Streams

Removed *L* from the wording of 13.13.2(9), since it is never used after it is defined. Also, the fonts of the wording were messed up.

Tucker notes 13.13.2(40/2) needs a comma before "or".

AI-216/16 Unchecked unions -- variant records with no run-time discriminant

The Note B.3(74) was deleted as it is no longer true.

AI-230/16 Generalized use of anonymous access types

Two Ada 95 AARM notes were corrected; AARM 7.3(7.d) and 7.5(2.c-d) assumed that access discriminants only are allowed on limited types.

AI-237/08 Finalization of task attributes

Corrected C.7.2(15.1) to make sense.

AI-247/03 Alignment of composite types

The note 13.3(37) depends on the deleted paragraph; it was deleted, too.

AI-248/12 Directory operations

Search_Type needs to say that it needs finalization. Added additional text to the first note at John's suggestion.

Bob asks that "requires finalization" to be added to the index. This isn't in the index because the term is "needs finalization", and "needs finalization" is indexed.

The Implementation Advice on Rename was updated so it was possible to follow. The group doesn't like the rewrite, either. Tucker suggests: "Rename should be supported at least when both New_Name and Old_Name are simple names and New_Name does not identify an existing external file."

AI-251/19 Abstract Interfaces to provide multiple inheritance

Adam Beneschan has complained in e-mail that the wording of 4.6 allows conversions between any two types. Steve Baird agrees with Adam. He also says that there is a problem if there are two unrelated types doing conversions. Tucker doesn't believe that can happen.

Steve is asked to write an example:

```
      type TT is tagged null record;
      type Ifc is interface;
      type Ref is access all TT'Class;
      procedure Foo (X : Ifc'Class) is ... end Foo;
      Ptr : Ref := ...;
   begin
      if Ptr.all in Ifc'Class then
         Foo(Ifc'Class(Ptr.all));
      end if;
   end;
```

You want this to work, as some descendant of TT might inherit from Ifc.

So 4.6(23.1) should be moved down a lot. If there is a common ancestor, it works fine. No, it doesn't; the wording is really wrong.

Tucker suggests adding a third category of conversions. But that doesn't work, because once you drop into one of these categories, you don't get out.

So duplicate the rules for interface conversions into each grouping.

Tucker still tries to avoid it, by saying it in the header of 4.6(21):

"...operand type, or both types are class-wide, then:"

Similarly in 4.6(24).

Add "either" in 4.6(21) to handle Adam's concern.

Tucker then suggests to reword 4.6(24):

"If there is no type that is the ancestor of both the target type and the operand type, and they are not both class-wide types, then conversion is permitted only in the following cases:"

Bob says that then the wording of the bullets would have to be reworded.

Much discussion ensues. We end up with the following for 4.6(24):

"If there is no type that is the ancestor of both the target type and the operand type, and they are not both class-wide types, one of the following rules shall apply:"

The rules for this part are mutually exclusive. Pascal suggests that we use the same wording for 4.6(21). There we need “at least one”, because the rules aren’t mutually exclusive.

“If there is a type that is an ancestor of both the target type and the operand type, or both types are class-wide types, then at least one of the following rules shall apply:”

Steve Baird worries about another issue, but there is no problem.

Approve correction to AI-251: 9-0-2.

AI-266-2/10 Task termination procedure

The title and introduction of C.7 was changed to reflect the addition of C.7.3. The package was made preelaborated. The wording was changed a lot by a variety of people. We decide to let the reviewers of Annex C check the detailed wording changes.

AI-270/06 Stream size item control

Recommended level of support is usually formatted as a list of bullets.

The wording needed a requirement for the number of stream elements written by Read/Write to match that specified by T'Stream_Size.

The Implementation Advice (13.13.2(17)) was moved to the attribute Stream_Size and now specifies the default value of T'Stream_Size.

Stream_Size was changed to be a representation attribute; all possible operational attributes are expected to be supported, but we don’t want to require implementations to be able to stream 480-bit integers.

Added Stream_Size to the list in 13.3(77). Erhard and Pascal would like to alphabetize this list of attributes. The group agrees.

AI-285/15 Support for 16-bit and 32-bit characters

Add char16_nul and char32_nul to the Implementation Advice of B.3(62.1).

B.3(39) needed to reference ISO/IEC TR 19769:2004, and this document was added to 1.2.

AI-287/11 Limited aggregates allowed

The reference in 9.1(21) was made more specific (to C.7.1). Tucker complains about the wording of 9.1(21/2). Randy complains that these notes were not reworded by anyone, and he just had to muddle them himself. Tucker will look at all of the notes that AI-287 needed to change (or maybe should have) and provide rewording if needed.

13.4(6) was changed to prevent the use of <> in enumeration_representation_clauses. Tucker had suggested a bizarre meaning for <> in these clauses. The group was impressed by his inventiveness; but <> is still illegal here.

The rule allowing an **others** => <> to match no components in a record aggregate (which was in AI-389) was moved here. Erhard complains that part of the AI that he hated so much is coming back. Randy explains that he had put the rule into AI-389 because it was easier at the time, but he considered it a bug fix to this AI. It’s odd to require that **others** => match at least one component; that’s not the rule for array aggregates or for case statements. It was necessary for expressions in record aggregates in order to have an expected type to resolve to, but <> has no such requirements. After discussion, the group agrees with Randy.

AI-287/12 Limited aggregates allowed

On Monday, we look at Tucker's update. He reworded the notes in question: 7.3.1(12), 9.1(21), and 9.4(23).

Pascal is concerned about "precludes" the use of predefined equality operators. He is convinced that this is a note, it isn't that important to be exactly correct.

Approve corrections with changes: 10-0-0.

AI-291-2/04 By-reference types and the recommended level of support for representation items

The paragraph numbers were reorganized to make better use of existing text. Dropped a stray *X* from 13.3(28).

The paragraphs added after 13.3(25) were moved, as they have "need not" wording, which is Recommended Level of Support wording, not normative wording.

13.3(35.2/2) should not have a bullet. Tucker disagrees, and Randy withdraws his concern.

There is an extra word in 13.3(35.1/2): "...alignments [that] supported...".

AI-296/10 Vector and matrix operations

The error analysis formulas were corrected.

AI-297/12 Timing events

The AI reflects the split of D.13 into two subclauses. The wording was changed a lot by a variety of people. We decide to let the reviewers of Annex D check the detailed wording changes.

AI-301/12 Operations on language-defined string types

Corrected the presentation of File_Type in some of the specifications.

AI-302-3/12 Container library

Added Preelaborable_Initialization to private types in the containers packages.

A number of errors in the examples in the discussion were corrected.

A cut-and-paste error has "... or designated an element..." in 7 places; which should be "designates".

Tamper-with-cursors for vectors included Delete_Last by mistake, only in the AI (not the AARM).

In Delete for Lists, changed "less" to "fewer", and improved the parenthetical remark.

Approve corrections: 6-0-4.

Open issues for containers:

Question 1) Matt noted that Delete, Delete_First, and Delete_Last have different behavior when called on an empty vector. He proposes that Delete allow the value Last+1, in the same way that Insert does.

Allowing Last+1 in Delete is OK with Pascal. No one else objects. So we adopt Matt's solution.

2) Matt had shown an example where the parameter of Query_Element could be clobbered by an operation in an outer scope. This was grist in a long running discussion about the tamper-with-elements rules; Matt thought that the consequences were counter-intuitive.

Randy said that the example seems obscure, and requires explicit up-level accesses. We could prevent with an extra check. Tucker says that preventing that case is too much.

Tucker goes on to wonder why Update_Element is included with tamper-with-elements. Randy says that it was intended to be similar to Replace_Element. The tamper-with-element checks are intended to prevent problems with indefinite containers deallocating memory that has been bound as an **in out** parameter. It also prevents data loss if a by-copy parameter is modified by multiple calls (only the last change will be preserved). Tucker points out that the dangerous case cannot happen for Update_Element, as it cannot change the discriminants for an indefinite container. After discussion, we decide that Update_Element does not tamper (with elements or cursors), but it still checks that tampering does not occur in Process.

3) Matt had wondered if "the container is not modified" after the failure of an allocation for Insert of a List was intended for Count > 1. That would have some implementation overhead.

Pascal says that it is hard for the user to know what to do if some random number of elements is added to the list. All or nothing is much easier to handle. Therefore, we will not change the wording; the implementation will need to allocate on a temporary list, or restore the state on an exception.

We should add Implementation Advice to not lose memory in this case. "If an exception is propagated from one of these routines, no storage should be lost." That advice should be added to all of the containers.

4) Matt is concerned that the Program_Error check for an unsorted list in Generic_Merge is O(N), and requires walking the list even once the merge is completed. Randy speculates that perhaps it would be better to just say the result order is unspecified if the input lists aren't sorted. A bounded error is another possibility.

Pascal feels very strongly that for a Generic_Merge that the result is sorted. He believes it should not be possible to successfully complete a merge without getting a sorted result.

Several people wonder about the value of the check. It is O(N) to make this check, and this is expensive, similar in magnitude to the merge itself. Steve Michell suggests that one could (always) keep a bit that the list is sorted. But that isn't possible as that would require the element ordering operator to be available on the entire container, as opposed to just the sort.

If we drop the requirement for a check, we need a need a function Is_Sorted so the user can check the predicate (or postcondition). That would have to be a generic. Perhaps we should put all three operations in a package Generic_Sorting.

Tucker would like to have a Merge procedure on Vectors. He says that the operation wouldn't be efficient, but it is well-defined.

So we have a package Generic_Sorting, that contains routines Is_Sorted, Merge, and Sort, for both Vectors and Lists. Merge has a bounded error; it returns Target with all of the elements in an unspecified order; then Program_Error may be raised. Pascal still does not like that.

5) The three- and four-parameter versions of Splice are inconsistent in their handling of Position => No_Element. Also, Matt thinks that the fourth parameter of the four-parameter Splice ought to have mode **in out**, because the cursor is changing the list container that it lives in, and that requires an update (the container is referenced by the cursor).

We agree with Matt that Position should be **in out**. Should the value returned in Position be No_Cursor or the position of the element in the new list? Tucker says that updating the cursor seems useful. Randy says that the cursor designates the element, not specifically the list, so changing the container matches the semantics.

Steve Michell says that doing so would make it harder to iterate using such a cursor. No, it doesn't; iterating directly would be a bounded error (because the cursor would be in the wrong container). Tucker suggests a version of Splice

where the cursor moved to the next element to facilitate iteration. But that seems like a tricky definition, and we don't know the direction the user wants to iterate.

Tucker suggests moving the second Splice to the third position on this list; it is a specialization of the third one. Make the third parameter of second one (now third one) to be **in out**; it has to match the more general routine.

Pascal wonders if this third operation is Splice at all; he thinks it does something different (shuffling elements within a single list). Others think it is similar enough. There doesn't seem to be consensus for a change.

Tucker would prefer that the Target parameter name be changed to Container. This second (now third) Splice should be defined in terms of the third (now second) Splice. "Equivalent to Splice (Target => Container, Before => Before, Source => Container, Position => Position);". Doing this automatically fixes the first part of question as well.

The description of the first Splice should avoid the awkward phrase "... Program_Error is propagated unless..."; it should start with "If"; the first sentence should read the same as the second sentence of the third (now second) Splice.

Approve corrections to AI: 5-1-5. Pascal opposes the merge change; he is happy with the other changes.

AI-305/09 New pragma and additional restriction identifiers for real-time systems

No_Protected_Type_Allocators was updated to cover subcomponents.

The missing lead-in was added in front of D.7(15). The lead-in should be singular, because there is only one restriction that it introduces.

AI-307/11 Execution-time clocks

As with AI-297, the wording was changed a lot by a variety of people, and we'll let the reviewers of Annex D check the detailed wording changes.

AI-327/08 Dynamic ceiling priorities

The lead-in for the attribute Priority was changed to be more similar to those of other attributes. It should say "an enclosing protected object". Randy complains that would be different than E'Caller, which has a similar usage restriction.

Change the last sentence to "A reference to this attribute shall appear only within the body of P." Drop the word "enclosing" from the first sentence, because objects don't enclose, and the legality rule covers it anyway.

AI-337/03 Applicability of C interfacing advice to private types

Changed "type's full view" to "full view of the type". Alan shouldn't be the only one required to remove possessives.

AI-344/11 Allow nested type extensions

The last sentence of 3.9.1(3) was marked to be deleted, but we certainly don't want to delete this requirement.

Erhard asks about the wording of 3.9.1(3.b). Tucker suggests adding "Ada 95 required the record extension to be the same level as the parent type. Now we use...on class-wide allocators..."

Gary had sent a question about this AI via e-mail. He is asked to show an example of the problem.

```
package Pkg is
   type TT is tagged ...;
   type ATT is access all TT'Class;
end Pkg;

with Pkg;
procedure Proc is
   type NTT is new Pkg.TT with ...;

   generic
      type Acct is access all Pkg.TT'Class;
   package Gen is
   private
   end Gen;

   package body Gen is
   begin
      ... := new NTT; --(1)
   end Gen;

end Proc;
```

The allocator at (1) would be illegal if it was outside of the generic body; it's caught by the static check. But there is no dynamic check, and we're in a generic body, so legality rules aren't enforced at instantiation time. AI-344 should be fixed to include a dynamic check.

Gary suggests adding "...the accessibility level of the type determined by the subtype_indication, or by the tag of the value of the qualified_expression, is not ..." to 4.8(7). Gary asks if we should be rechecking the static check in the private part. There is no hole that needs to be fixed, and it seems not worth it to do that.

Approve new correction to AI-344: 10-0-1. [Editor's note: This correction is in version /11, created after the meeting.]

AI-345/08 Protected and task interfaces

Corrected 3.9(2) to include all of the new kinds of tagged types. Add a missing word in the AARM: "...is {a} tagged type..." (this is correct in the AI).

Corrected 3.9.4(1) (this was wrong only in the AI and Amendment, not the AARM).

AI-354/08 Group execution-time budgets

As with AI-297 and AI-307, the wording was changed a lot by a variety of people, and we'll let the reviewers of Annex D check the detailed wording changes.

AI-355/08 Priority Specific Dispatching including Round Robin

Reused the number of D.2.2(18). Shortened the title of D.2.2.

AI-357/09 Support for Deadlines and Earliest Deadline First Scheduling

Moved the Static Semantics section to its proper position.

Added "In these cases" after the first bulleted list.

AI-360/07 Types that need finalization

Removed redundant cross-reference in D.7(4).

AI-376/02 Interfaces.C works for C++ as well

Added a reference to the C++ standard in 1.2.

AI-385/04 Stand-alone objects of anonymous access types

We added wording to allow deferred constants of anonymous access types.

Someone asks why anonymous arrays can't be allowed as a deferred constant. They potentially have elaboration issues, which anonymous access types don't have.

AI-386/03 Further functions returning Time_Span values

Lined up the declarations properly.

AI-394/03 Redundant Restriction Identifiers and completing Ravenscar definition

No_Dynamic_Priorities should not be moved to Annex J, as AI-327 extends its definition.

AI-397/03 Conformance and overriding for entries and protected operations

Randy says, and Pascal concurs, that it is unusual to not allow **not overriding** on entry families. That will make them look like they mistakenly omitted the indicators. Tucker disagrees, saying that there is no value to saying something that always has to be true.

We take a straw poll: Allow **not overriding** on entry families: 2-7-1. The syntax rule should be replaced.

Randy is concerned that we have conflicting rules for **overriding** on protected subprograms. These are syntaxtically subprograms, so both the rules in 8.3 and the rules in 9.4 would apply.

Pascal says that that there is no conflict, protected subprograms implement while subprograms override. Randy says that since a protected subprogram never overrides, it would require **not overriding** (by the rules in 8.3); but if it implements something, it would require **overriding** (by the rules in 9.4). That won't work.

Randy wonders if these rules shouldn't be moved to 8.3. Had that been done, this problem would have been obvious. We talk about that for a while, and decide not to move the rules. Add an exception to 8.3(26) to exclude protected subprograms.

Erhard worries that conventions of implemented subprograms don't match, but subtype conformance would require them to match. Pascal says that was fixed in Paris, in this AI in 6.3.1(24.1).

Approve latter correction: 11-0-0.

AI-399/02 Single task and protected objects implementing interfaces

Corrected the indentation of the syntax.

AI-400/03 Wide_ and Wide_Wide_ images of identifiers

Clause C.5 on pragma Discard_Names talked about the original versions of these functions.

AI-401/03 Terminology for interfaces

Moved the italics for “parent type” from 3.4(1) to 3.4(3).

Removed the possessive from 3.4(27).

AI-404/04 Not null and all in access parameters and types

The wording of the new paragraph after 3.9.2(11) was updated to reflect that controlling access results are not always null excluding.

AI-407/04 Terminology and semantics of prefixed views

Changed the comment on the example in 4.1.3(17) to use the term “prefixed view”.

AI-409/02 Conformance and overload resolution related to anon access types

6.3.1(13.1) and 6.3.1(15) needed “access result”.

AI-412/04 Subtypes of incomplete types; renamings of limited views

The text added following 8.5.3(3) was rewritten.

Added library_unit_renaming_declaration to 10.1.2(18/2), as it would be impossible to use the name of such a library unit in any context.

AI-414/02 pragma No_Return for overriding procedures

Deleted the last paragraph of this subclause as it is covered by B.1(38.1) (from AI-320).

Changed the syntax to use *procedure*_local_name to match pragma Pack and other similar uses.

Detailed Review of Amendment AIs

AI-302-4/00 Container library (mail container)

This just holds the mail; the mail has been addressed.

No action: 11-0-0.

AI-381/04 New Restrictions identifier No_Dependence

There was a consensus in e-mail to change this AI; Randy rewrote it.

In the !wording, “language-defined” should be removed.

Tucker says that the syntax should not have been dropped; the name is necessary. Randy doesn't buy it, saying that expression includes name, but he is overruled. Revert to the previous syntax change.

John would like the "sequence of identifiers" part of the AARM note dropped, because it is strictly more than a sequence of identifiers (it also includes dots). Randy groans that the pedants are taking over the world.

It is thought that the point about any name being allowed needs to be made normatively. Add at the end of the legality rule, the "..., but need not denote a unit present in the environment."

Drop the first AARM note.

Pascal worries that typos in unit names would be missed. That's the trade-off; you lose the checking that you spelled the name of the unit right. But to keep the language-defined only rule, we would have to define what is a language-defined unit; Bob has pointed out that's not as easy as it appears. Moreover, this pragma would be very useful to avoid dependence on implementation-defined units (package GNAT, for instance), and it would be a disservice to Ada users to make them use some other way (or have no way) to specify that they want to avoid such units. An implementation could have a warning if no such unit exists. Finally, this is not a pragma that will be frequently written; it will be written once and probably unchanged for the life of the project. Taking extra care with a rarely used pragma seems like an acceptable burden given the additional utility gained.

Approve AI with changes: 10-1-0. Pascal still wants a check for typos.

AI-395/06 Various clarifications regarding 16- and 32-bit characters

Most of the items were approved in Paris; we'll only look at the ones that were not.

Item 2: 6.1(10) is on operator symbols.

Steve Michell suggests that the last statement should say ", but not otherwise." or something like that to say that the characters are not allowed in operators.

Tucker, looking at 2.1, would like to move other_format and punctuation_connector before separator_space, to put all of the graphic characters together. This will be handled as an approved correction to AI-285.

AARM syntax 2.3(3.1/2) is missing punctuation_connector in the syntax.

Tucker would like to move some of the text from 6.1 to 2.6. He will write that.

Bob doesn't understand why we need to change 6.1(10). We give him an education about Unicode identifiers and reserved words.

Gary is worried about "operator symbol" in the AARM note. He thinks it is confusing. Tucker will reword this, too.

Item 9: Steve Baird notes a typo in the !proposal; "operation{s} on Wide_ and ...". Above it, the second wide is not capitalized.

Tucker would prefer that the specifications of the routines be removed as well. Put the names in a comment so people will know they exist.

Approve item 9 with changes: 5-0-6.

On Sunday, Tucker asks if his wording is OK. No one has seen it (it's not in the mail), so we make him write it.

"The sequence of characters in an operator_symbol shall form a reserved word, a delimiter, or compound delimiter that corresponds to one of the six classes of operators defined in clause 4.5."

Note that "form" is used as a technical term here; 2.2(1) uses "form" in this sense.

Pascal asks if it is clear that no spaces are allowed. Tucker replies that this doesn't say that an operator symbol is a bunch of junk that happens to include a reserved word or delimiter. That causes general laughter.

Pascal asks if a reserved word allows other_format at the end. We read the wording in the AARM, and it is clear.

Revise the AARM note here:

“AARM Note: The “sequence of characters” of the string literal of the operator is a technical term, and does not include the surrounding quote characters. As defined in 2.2, lexical elements are “formed” from a sequence of characters. Spaces are not allowed, and upper and lower case is not significant. See 2.2 and 2.9 for rules related to the use of other_format characters in delimiters and reserved words.”

Approve AI with changes: 10-0-1.

AI-416/05 Access results, accessibility, and return statements

Tucker explains the changes to the AI.

Nested things get the same accessibility level in extensions as in the parent.

```
type Foo is tagged
   record
      X : access T;
   end record;

type Bar is new Foo with
   record
      Y : access T;
   end record;
```

The accessibility of X is the same in Bar as in Foo, even if Bar is more nested than Foo.

Erhard notes that the parameter composition of constructors doesn’t work:

```
type T (A : access Integer) is ...;

function F (A : access Integer) return T is
begin
   return T’(A=>A); -- Program_Error from result accessibility check.
end F;

X : T := F(G(7)); -- Raises Program_Error.

X : T := T'(A => G(7)); -- OK.
```

The nested call G(7) would have a very short lifetime by the 3.10.2(10) of this AI, so the result accessibility check would fail in the return statement (the function call having the accessibility of the declared object). That’s necessary, because the temporary object will disappear as soon as the object declaration is finished. On the other hand, the hand-expansion of the call as an aggregate is fine, because G(7) would get the accessibility of the result object.

Tucker explains that the problem is the evaluation of G(7) is too soon; we’d need lazy evaluation to allow this. Or we'd have to deter finalization/deallocation of all parameters to functions used in initializing expressions until the declared object disappears. That’s clearly not acceptable. Erhard is satisfied by that explanation.

Tucker shows:

```
return Result : T := (  ) do
   -- Result is local here.
end return; -- Result changes to the accessibility of the result here.
```

Steve Baird complains that this would allow a local variable as a discriminant.

Tucker says that there is a separate accessibility check on the discriminants, see 6.5(20).

Examples of the effect of 3.10.2(12):

```
X : T (D => G'Access); -- Level of G.

X : T := (D => G'Access); -- Level of G.

X : T; -- Level of X.
```

Examples of the effect of 3.10.2(14:

```
subtype S is T (D => new Q); -- Allocator level of S.

X : T (D => new Q); -- Allocator level of X (the "joined at hip model").

X : S; -- Level of S, doesn't change.

type R is
   record
      C : T (D => new Q); -- Allocator level of R.
   end record;
```

Steve Baird says that the unchecked deallocation rule added by AI-162 is wrong. Perhaps we need a technical term for “joined at the hip”. Someone suggests co-terminus. Steve Baird says “coextensive”, which he claims is a real English word. Tucker will try to fix 13.11.2(17.1/2) by tomorrow.

There seems to be an issue when the level changes, as in a return statement; the “joined at the hip” object needs to change too, otherwise the discriminants would dangle. Tucker will look at that too.

We need 4.8(5) to catch cases like:

```
P : Acc_T := new T(D => G'Access); -- G has to outlive the Acc_T type.
```

4.8(7) is the corresponding runtime check.

6.5(5) is the same check for a return statement.

```
return T (D => G'Access);
```

Tucker now announces that he is feeling a bug coming on here. He writes the following on the whiteboard:

```
return T (D => new S(E => G'Access));
```

When you have a joined-at-the-hip guy, you need to check the accessibility all the way down. Another thing for Tucker to fix.

“Coterminous” (or some other technical term) is looking better by the minute.

Tucker now worries about the static class-wide rule. We write some examples:

```
function ... return T'Class is
   type NT is new T with ...;
   X : NT;
begin
   return X; -- Static check.
   return T'Class(X); -- Dynamic check.
```

That’s OK, and there are no problems.

Pascal is concerned that the return object is not declared. 3.3 needs some reference to return objects. Tucker suggests copying from allocators. That says “created”. So we need to say that in 6.5(5.5/2). Tucker is convinced that is all we need. He will reword.

In 6.5(22/2), move second “if” before the “while” that is: “...call if while elaborating ..., and it is determined...”

Steve Baird is worried that this wording doesn’t say the function body stops at this point. Tucker suggests saying that “the function body is abandoned”.

7.6.1(3): Steve Baird points out that subtype_declarations also should not be masters. Tucker wonders why we include declarations here at all. It seems that the expression covers it. We are not too sure, so we decide not to change this.

Accessibility is overridden in some cases, so there isn’t a problem with subtypes. Types, however, do use the default accessibility. Tucker will check this again, just to be sure.

John notes two consecutive spaces in the first sentence 9.2(3) of the AARM.

Steve Baird asks if in 6.5(22), “used” includes indirect uses. Add “...used directly or indirectly...”.

Approve intent of AI: 8-0-3.

AI-416/06 Access results, accessibility, and return statements

There is discussion about the parentheses that were added to 3.10.2(7); Tucker says the text is redundant because a declaration doesn’t elaborate itself.

The new wording in 3.10.2(14) defines “coextensions”. These mean what we’ve informally be calling a “joined-at-the-hip discriminant”.

Steve Baird asks whether the tasks of a coextension of a return object are activated before return (unlike the tasks directly in the object). We’ll look at that later.

Tucker has made the deallocation of coextensions at the same time as the enclosing object mandatory. It was just Implementation Advice previously, but that’s weird when finalization is required to happen.

Why is the term “coextension”, rather than “coterminus”? Tucker explains that he didn’t use “coterminus” because that is an adjective.

Gary says that the term bothers him a bit, because it makes him think of type extension, and this has nothing to do with type extension. John and Randy agree. Pascal says that coextension applies to objects; and type extension applies to types, so it isn’t confusing.

Tucker reads a definition of “coextension”: “has the same extent in time or space”. That does seem right.

Steve Baird worries that defining the storage pool allows doing some Unchecked_Deallocations portably. That doesn’t seem to be a problem (any more than the basic idea is), in fact it seems good.

We look back at Steve’s issue with task activation. Allocators always start at the same time. Is there a problem? Well, all these tasks would be running in a return object, and then the master would change when the object is returned. Changing the master of a running task is horrible! They have to get activated at the same time; it should work the same as finalizations.

So 9.2(2-4) need to talk about “...parts or coextensions...”.

We try to look at Adam Beneschan’s e-mail comment. It looks like we fixed that in Paris.

Approve AI with changes: 8-0-1.

At the very end of the meeting, Steve Baird asks about renaming a function call that includes a coextension. Tucker says that this is the same as passing a function call to a subprogram with the same lifetime as the renames.

Steve claims that you don't know the finalization until runtime, so you don't know statically when to finalize it. Tucker doesn't think this is any different than any other case; it has nothing to do with renames. Steve Baird says that this is different than Ada 95. Others are skeptical. Steve writes an example:

```
X : Integer := F.Sc'Length;
-- Finalize F here.

X : Integer := F.Fld.all;
-- Finalize F here.

X : Integer renames F.Fld;
-- Finalize F later.

X : Integer renames F.Discrim.all;
-- Don't know whether you can finalize F here, because you don't know if
-- the discriminant is a coextension.
```

Tucker says that everything involved in the renaming doesn't get finalized until the renaming goes out of scope. You don't do that piecemeal. So you always finalize F later in the last example above. Thus, there is no problem and no difference from Ada 95.

AI-417/01 Lower bound of functions in Ada.Tags and Ada.Exceptions

There is a discussion about the part of the wording that was not changed by this AI. Eventually, we decide to leave it as it is.

Approve AI: 10-0-1.

AI-418/02 Vector norm

Is this the correct notation for this operation? John replies that "abs" is used for the modulus of a complex number, and this is the same thing, so it should have the same name. This operation also uses the absolute value symbol in mathematical notation (vertical bars).

Typo in AARM Implementation note: "...that [is]{it} has accuracy requirements..." Drop the semicolon in this sentence.

Approve AI with changes: 10-0-1.

AI-419/02 Limitedness of derived types

The idea is that you can specify **limited** for a derived type.

Erhard is worried that 3.4(15) seems to apply to task types derived from an interface. This paragraph only applies to derived_type_declaration. This is wrong; the real definition is in 7.5 anyway. Remove this rule altogether, and move the AARM notes to 7.5.

We probably need a rule like 7.3(16) for task and protected types. Add that to 9.1 after (9.1) and 9.4 after (11).

Should tasks and protected types be derived typed? Derived types always have a parent type, and these don't have one. We could promote one of the interfaces to be the parent, but that would make one interface different than the others for no good reason.

Similarly, interfaces are not derived types; they also don't have a parent type.

We need to say that a derived_type_declaration declares a derived type, since that is where the wording is, because we now have things that are derived from that are not derived types. Add that to 3.4(1).

Tucker will craft wording for this (3.4(1), 9.1(9.1.5), 9.4(11)) for tomorrow.

We finally understand a good reason why interface types aren't derived types — derived types have parents, which interfaces don't have.

Typo in the last paragraph of the proposal: "limitness" should be "limitedness".

Erhard goes off the agenda and complains about having the syntax of interface_list in 3.4. We agree to move it to 3.9.4, and move the first sentence of 3.4(3/2) to 3.9.4(3). If we do that, we should have a cross-reference to 3.9.4 somewhere. Bob and Pascal would like to keep a redundant part of 3.4(3); that would hold the cross reference.

"The interface_list defines the progenitor types (see 3.9.4)."

Approve intent of AI: 9-0-2.

AI-419/03 Limitedness of derived types

Gary would like to drop "new" from 3.4(1). That seems fine.

Tucker noted a typo in the AARM, in 9.1(9.2/2): "...designating {t}he task type...".

Approve AI with changes: 11-0-0.

AI-420/01 Equality of anonymous access types

We can't have user-defined "=" for anonymous access types. This is similar to fixed-fixed multiplying operators, and the fix is similar. We also add a way to get to the predefined operator, because if we don't have it, it wouldn't be possible to write one of these user-defined "=" (how would you test against **null**?).

Change the title to include the Standard stuff. "User-defined equality of anonymous access types" is better, but not well-liked, but no one can suggest something better.

Typo: "...using {an} expanded name..."

Tucker suggests the subject of "Resolution of universal operators in package Standard".

Approve AI with changes: 8-0-3.

AI-421/01 Sequential activation and attachment

The first bullet should end with "...is deferred until all library units are elaborated."

Tasks are created, not elaborated (declarations are elaborated). So change "elaborated" to "created" in the first bullet.

There is discussion about whether interrupts should be attached before or after the activation of tasks. It's more predictable to have interrupts attached first.

Alan will create new wording by tomorrow.

Steve Baird wonders where the exception raised by a failed activation is handled. The original AI covers that. But the environment task is not terminated, it is completed. And all of this is the standard Ada semantics. Bob Duff says that we should just say that the place is one that could not have a handler, and everything else follows.

Pascal suggests saying "...is raised in at the beginning of the sequence_of_statements of the environment task (see 10.2)."

Steve Michell suggests adding a user note to say that the handlers in package bodies that otherwise would handle Tasking_Error do not.

AI-421/02 Sequential activation and attachment

Alan rewrote the paragraphs. But he dropped the list of bullets presentation, and this not well-liked.

The second paragraph should say “prior to calling the main program”. We previously said: “...is raised at the beginning of the sequence_of_statements of the environment task prior to calling the main program (see 10.2).”

In the note: Change to “the Tasking_Error exception”. Change the last part “unable to handle the Tasking_Error exception and it completes immediately.”

“If the task activation fails, then the environment task is unable to handle the Tasking_Error exception and completes immediately. In contrast, if the partition elaboration policy is Concurrent, then this exception could be handled within a library unit.”

John will try to write the words tonight.

AI-421/03 Sequential activation and attachment

Remove “then” from bullets 2 through 4.

Remove the first “in the environment task” from the next paragraph (it’s redundant).

Add a word to the note: “If any {deferred} task activation...”

In the paragraph following the bullets, reverse the sentences. Make them separate paragraphs.

Gary says the first bullet needs a hyphen in “library-level”. Always say “library-level tasks”, not “library tasks” (change the third bullet accordingly).

“...the same interr{r}upt are deferred{,} then the ...”

Approve AI with changes: 10-0-0.

AI-422/01 User data for timing events

John wonders why we don’t just define equality on these objects. Pascal says it is just the object addresses that matter.

Alan says that these were intended to be similar to bare interrupt handlers; no fancy semantics.

Tucker suggests adding **tagged** to the types and 'Class to the handlers. 'Class would interfere with the No_Dispatching restriction. Bob says that the definition of this restriction is silly ('Class doesn’t have anything directly to do with dispatching), but no one wants to try to change it. (A change would likely be incompatible.)

Just **tagged** alone would work. An explicit conversion in the handler would allow access to any added data. And doing so also makes it by-reference, so comparing 'Access of the objects would also work.

This should be added to Timers and Executions.Timers and group budgets.

Randy asks whether this should be processed as a correction. A correction would only require adding the new text to the places needed. That seems like that is preferable, and we will drop this AI.

No action for the AI: 11-0-0.

Approve intent of adding **tagged** to Timers, Execution_Time.Timers, and group budgets: 10-0-1. (We’ll approve the correction itself at the next meeting.)

AI-423/03 Renaming, null exclusion, and formal objects

Pascal explains what is wrong with the existing option.

Steve Baird blows our minds asking about what happens when you rename a generic **in out** parameter. That seems to be the same question as the AI is addressing.

Tucker says the point is to check that you are not lying when you say **not null**. If you don't know, then you can't allow it.

```
generic
   A : in out T;
package G is
   O : not null T renames A;
```

We could also go to requiring an exact match for **not null**.

We have required exact matching for formal access types. Pascal comments that a lot of rules assume that you do know, adding rules for "you don't know" cases seems weird.

So we think **not null** must never lie. It should be an "if and only if" rule.

Pascal will update the AI's wording tonight.

Approve intent of AI: 10-0-1.

On Sunday, Pascal announces that yesterday's decision doesn't work. It runs into problems with types that are not known to be access types, but actually are (that is, private types). Tucker tries to write an example:

```
generic
   type T is private;
   with procedure P (A : T);
   X : in out T;
procedure G is
```

What is the matching here? Does the **not null**ness have to match for P.A and X? That seems like it could cause trouble.

Someone asks if the problem is from renaming of formals:

```
generic
   type T1 is private;
   type T2 is new T1;
   C : T1;
package G1 is
   X : T2 renames T2(C); -- Legal?
end G1;
```

But this only is allowed for view conversions, and T2(C) is a value conversion. So this isn't a problem.

Tucker suggests that exact matching is OK. Someone shows an example:

```
type A is access ...;
procedure Q (P : not null A);
Obj : not null A;

procedure Goof is new G (T => A, P => Q, X => Obj);
            -- Illegal by exact rule.
```

Bob says that declaring a subtype works to fix this:

```
subtype AS is not null A;
Obj2 : AS;

procedure OK is new G (T => AS, P => Q, X => Obj2);
```

But that doesn't seem to extend well. Tucker suggests going back to the no-lying rule.

Another idea is to do exact matching if you know it is an access type, and anything goes otherwise. That doesn't seem to work because of types that gain information (in private parts and bodies).

Steve Baird asks about renames in generic bodies. That can be handled with an assume-the-worst rule in generic bodies. This case is too weird to worry about capability lost by conservative checking.

Pascal will again try to come up with wording tonight.

Approve intent of AI: 11-0-0.

AI-423/04 Renaming, null exclusion, and formal objects

Tucker suggests changing 8.5.1(4), in first bullet:

"...shall denote a..." , "...unit only if..."

The generic body part of this is wrong, it should read:

"if the object_renaming_declaration occurs within the body of a generic formal unit, and the object name denotes a generic formal object of that generic unit, then the declaration of that formal object shall have a null_exclusion;"

We need to change all of these the same way.

Approve AI with changes: 9-0-1.

AI-427/02 Default parameters and Calendar operations

John explains his issues and solutions. The summary for #4 is wrong (it reflects a previous version of the AI).

Tucker would prefer to have the bounds for Leap_Seconds_Count be 2000. Pascal would prefer 1612, which is the actual maximum number. Someone jokes 2006. 1612 is liked, but the derivation of the number is mysterious. Finally, after way too much discussion, we decide to make no change to the AI.

In the AARM (not the AI), in 9.6.1(4.a), "are than" should be "are more than"

Remove the first sentence of the !discussion; we don't care if the French are posh.

The wording for the new Year, Month, and Day functions is missing. The wording for Seconds and Sub_Seconds needs to be adjusted to remove the mention of the time zone. The specifications of the various Split routines aren't modified in the subprogram descriptions.

Tucker would prefer saying Time_Offset as 28*60 instead of 1680.

Approve AI with changes: 11-0-0.

AI-428/01 Input-output for bounded strings

The with clause is wrong, the last part is a nested unit, so it should be "**with** Ada.Bounded;".

Switch the order of the clauses for Unbounded and Bounded (Bounded should be A.10.11, Unbounded A.10.12).

Delete the second paragraph of the discussion (we don't need to discuss "Poor old John" in an AI).

Approve AI with changes: 10-0-1.

AI-430/00 Conventions of inherited subprograms

[Editor's note: This AI was assigned a number after the meeting.]

Steve Baird has issues with 3.9.2(10/1); he thinks that there are holes and confusions regarding interfaces. He shows an example of the confusion:

```
package P1 is
   type T is tagged ...
   procedure P (X : T);
   pragma Convention (C, P);
end P1;

package P2 is
   type TT is new P1.T;
   procedure P (X : TT);
   pragma Convention (Ada, P);
end P2;
```

Tucker says that the pragma is illegal here. Steve Baird says that 3.9.2(10) says that the convention is inherited; it doesn't seem to allow changing it. A plausible but unfriendly reading is that the convention pragma is ignored.

We should add the word "default" to this wording in 3.9.2(10). "The default convention...".

We start worrying about inheriting different conventions from interfaces:

```
type Int1 is interface;
procedure Foo ( I : in Int1) is abstract;
pragma Convention (Ada, Foo);

type Int2 is interface;
procedure Foo (I : in Int2) is null;
pragma Convention (C, Foo);

type T is tagged Int1 and Int2 ...;
procedure Foo (O : in T); -- Illegal (see below)
--inherit procedure Foo (I : in T); -- Convention Ada.
--inherit procedure Foo (I : in T); -- Convention C.
```

If the inherited subprograms are overridden, the subtype conformance requirement of 3.9.2(10) requires matching the convention. If they are not explicitly overridden, the rules in 8.3 say either that one overrides the other (triggering the subtype conformance rules) or full conformance is required.

Certainly, however, 3.9.2(10) needs "progenitor or parent type", so interfaces participate in these rules.

"The convention of an inherited dispatching operation is the convention of the corresponding primitive operation of the parent or progenitor type. The default convention of a dispatching operation that overrides an inherited primitive operation is the convention of the inherited operation; if there are multiple such conventions, then the overriding is illegal."

Bob and Erhard argue about the singular/plural mixing. Switch to all plural.

"The convention of an inherited dispatching operation is the convention of the corresponding primitive operation of the parent or progenitor type. The default convention of a dispatching operation that overrides an inherited primitive operation is the convention of the inherited operation; if the operation overrides multiple inherited operations, then they shall all have the same convention."

For “implemented by” case; since there is no overriding, the interfaces requires full conformance by 8.3. That’s required because we need to know the parameter name for the wrappers, so it doesn’t make an additional problem or undue restriction.

Approve AI with changes: 9-0-0.

Detailed Review of Regular AIs

AI-403/02 Preelaboration checks and formal objects

The wording section should start with “Replace 10.2.1(10) by”.

The AARM note should say “...preelaboration rule{s}.”.

Approve AI with changes: 11-0-0.

AI-424/01 List of language-defined units

Fix extra @ in the table of packages in the AARM (@@em).

Approve AI with change: 11-0-0.

AI-425/01 Organization of Annex M

Tucker asks that the first section be called “Specific Documentation Requirements”.

The introduction to M.2 uses “must”. Yes, but this is wording from Ada 95 and not normative here.

Take out all of the “shall be documented”s from M.1; usually you’ll read these as a group and they are too redundant.

Approve AI with changes: 8-1-2. Erhard would prefer that the introduction be changed; it says that the annexes need to be documented for all implementations, and he disagrees with that.

AI-426/01 Language-defined routines returning abnormal and invalid values

Randy tries to explain the problems he ran into and his fixes. The most important issue is that John’s example — the reason behind AI-167 — is unfixed.

John writes his example:

```
B : Byte;
for B'Address use 8#100#;
S : Status;

   ...

S := Byte_to_Status(B); -- An instance of Unchecked_Conversion.
if not S'Valid then
   ...
```

He expects this example to work, without raising an exception. Randy points out that is impossible to guarantee with the language as defined, because 13.9.1(9) applies. Thus, Program_Error or Constraint_Error can be raised.

Randy also thinks that there are other operations like T'Input and T'Read which have similar issues, and they ought to be handled the same way. He proposes some Implementation Requirements to make this happen.

Steve Baird explains that their implementation assumes that values are always valid. They forcibly load objects with valid values when necessary.

Pascal says that he expects that the bounded error would apply, and thus the programmer would have to check the exceptions. Randy notes that an implementation which makes the checks will avoid future surprises, unlike one that takes advantage of 13.9.1(12).

Tucker says that if the result of an Unchecked_Conversion instance is invalid, the compiler doesn't need to check if the ranges match. What does "match" mean here? "Statically match" seems to be meant; but note that there is no language rule which ever says checks aren't performed (outside of pragma Suppress) — any check that is omitted is an as-if optimization.

Tucker says that there is Implementation Advice that there is no check for Unchecked_Conversion. But that only applies to the instance of Unchecked_Conversion itself, not to the subsequent assignment. And that advice is that there are no "unnecessary" checks; it's clear that what is unnecessary to one implementation may be necessary to another. And it's advice anyway, so it doesn't have to be followed. So, if you want to be completely portable, you have to have an exception handler for Constraint_Error and Program_Error.

There seems to be agreement with that. We won't try to define cases where there must be no checks, so many of Randy's changes can be dropped.

There also should be a note near 'Valid that to be completely portable, you need to handle Constraint_Error and Program_Error because of the bounded error. Tucker will write this note.

We decide to drop the 13.9.1(8), 13.9.1(9), 13.9.1(11), and 13.9.1(12) changes.

[At this point, we left on a historical tour of Lexington and Concord (see Welcome), along with lunch. Given the exciting topic, it's surprising we ever came back.]

Typo in 13.9(11) AARM Note "...should [be] never be...".

```
S : Status;
for S'Address use 8#100#;
pragma Import (S);
pragma Volatile(S);

if not S'Valid then
    ...
end if;
```

Bob would like to say something about imported objects, as in the above example. "It a programmer's responsibility that an imported object starts and stays in a normal state."

Tucker suggests: "For an imported object, it is a programmer's responsibility to ensure that the object remains in a normal state." Add that after 13.9.1(6).

Steve Baird asks about exported objects. They would have to be declared volatile to change asynchronously (otherwise, they could only change when Ada code touches them, or the Ada code calls a foreign routine). The problem is with a volatile object, not with exported objects.

There is concern about the implementation-defined case for Unchecked_Conversion; Pascal would like a stronger requirement. But we can't define the bits returned, so it has to be implementation-defined. We decide to drop the "in particular" to make it clear that these always are possibilities; it's the bits that are implementation-defined. Bob says that the "the result of the function is implementation-defined" is how to talk about just the bits of the result.

Thus, the wording for 13.9(11) will be: "Otherwise, if the result type is scalar, the result of the function is implementation-defined, and can have an invalid representation (see 13.9.1). If the result type is non-scalar, the effect is implementation-defined, and in particular, the result can be abnormal (see 13.9.1).".

On Monday, Tucker provides the note wording.

21 The Valid attribute may be used to check the result of calling an instance of Unchecked_Conversion (see 13.9) if there is a possibility the result might be invalid. However, an exception handler should also be provided because implementations are permitted to raise Constraint_Error or Program_Error if they detect the use of an invalid representation (see 13.9.1).

Someone suggests inserting “that” in the above sentence: “...possibility {that} the...”.

Approve AI with changes: 8-0-2.

AI-429/01 Representation of minus signs

John explains the typography of manual is wrong — minus signs are too short.

Approve AI: 11-0-0.

Randy asks about John’s requests to make exponents larger in the various documents. After discussion, the conclusion is that they are fine in the printed (PDF) and HTML versions, so there is no issue.

AI-431/00 Conversions to remote access-to-subprogram types

[Editor’s note: This AI was assigned a number after the meeting.]

A question on Ada comment asks if local access-to-subprogram types should be convertible to remote access-to-subprogram types. There doesn’t appear to be a rule forbidding it.

Tucker says that T.**all**'Access would allow the conversion. We usually make this and a type conversion act the same. To make that this illegal, we’d have to make 'Access illegal in this case.

Tucker then reverses position and says that remote access-to-subprogram would not allow this, because the prefix of 'Access must statically denote a remote subprogram (by E.2.2(12)), and T.**all** does not statically denote anything.

Tucker suggests changing E.2.2(11) to say “...to or from another (subtype-conformant) remote access...”.

Approve AI to be constructed: 9-0-1.

AI-432/00 Out of range values in Ada.Real_Time

[Editor’s note: This AI was assigned a number after the meeting.]

Adam Beneschan asked on Ada-Comment what happens if the value doesn’t fit into a Time_Span in To_Time_Span. D.8(24) should say something about what happens if the value doesn’t fit in the type.

Pascal notes that this sort of issue affects all of the routines in Ada.Real_Time. We don’t know for sure that Duration has the larger range. He would like to add this as a blanket requirement, to D.8(27).

“A function that returns a value of type Time_Span propagates Constraint_Error if the result value is less than Time_Span_First or if the result value is greater than Time_Span_Last.”

Tucker says that doesn’t work, because there are no values outside of the range for Time_Span; this is not a numeric type.

We need to describe this in terms of time units.

Tucker will develop wording for this. We’ll need a full regular AI.

Approve intent of AI: 11-0-0.

Later on Monday, Tucker describes the wording for this AI. He modifies D.8(24) and (26).

The first two sentences of D.8(24) are unchanged.

The function To_Duration converts the value TS to a value of type Duration. Similarly, the function To_Time_Span converts the value D to a value of type Time_Span. For To_Duration, the result is rounded to the nearest value of type Duration (away from zero if exactly halfway between two values). If the result is outside the range of Duration, Constraint_Error is raised. For To_Time_Span, the value of D is first rounded to the nearest integral multiple of Time_Unit, away from zero if exactly halfway between two multiples. If the rounded value is outside the range of Time_Span, Constraint_Error is raised. Otherwise, the value is converted to the type Time_Span.

D.8(26):

The functions Nanoseconds, Microseconds, and Milliseconds, Seconds, and Minutes convert the input parameter to a value of the type Time_Span. NS, US, MS, S, and M are interpreted as a number of nanoseconds, microseconds, and milliseconds, seconds, and minutes respectively. The input parameter is first converted to seconds and rounded to the nearest integral multiple of Time_Unit, away from zero if exactly halfway between two multiples. If the rounded value is outside the range of Time_Span, Constraint_Error is raised. Otherwise, the rounded value is converted to the type Time_Span.

Approve the “Wonderful Wording of Tucker” (and the AI): 10-0-0.

Miscellaneous Topics

[Editor’s note: Since we had some extra time at the meeting, we asked members if they had any issues on the drafts that ought to be discussed. These issues are covered here when they didn’t lead directly to a correction or an AI.]

—

Pascal asks that we add the phrase “excludes the null value” to the index. [Editor’s note: after the meeting, it appears that this is not a technical term, and that we use different words in different parts of the manual. The terminology needed to be corrected to be more uniform, and the term “excludes null” was chosen. This correction will be reviewed at York.]

—

Pascal asks what the wording “apply in this Annex” means in Annex H. This means “supported by implementations that support this Annex”. Pascal suggests that perhaps the wording should be “are relevant in this Annex”. This is mostly a problem in H.4.

Paragraph H.4(2) doesn’t seem to be Static Semantics; it should be an Implementation Requirement. Get rid of the whole paragraph, and make it an IR before (23). “An implementation of this Annex shall support the following restrictions defined in D.7:”, then copy paragraph 2, along with the notes.

Pascal will write wording as a correction to AI-394.

—

Pascal had expressed concern about conflicts between a prefixed view of a primitive operation and a protected operation. Randy wrote an example and concluded that there was no problem. Steve Baird notes that the second two procedures in the examples in the e-mail are missing the Foo parameters. Randy shows the full, corrected, example:

```
type Int is protected interface;
procedure Bar (Baz : in out Int; I : in Integer); -- (A)

protected type Foo and Int is
   procedure Bar (I : in Integer); --(1)
end Foo;

procedure Bar (Baz : in out Foo; F : in Float); -- (2)
procedure Bar (Baz : in out Foo; I : in Integer); -- (3)
```

```
F : Float;
I : Integer;
Blab : Foo;

Blab.Bar (I); -- Protected call to (1);
              -- or a prefixed view call to (3). Error?

Blab.Bar (F); -- Prefixed call to (2); should this resolve?
```

Without the declaration (A), the declarations are legal. What about the calls? This is not incompatible, because you need the interface to trigger the difference. Randy thinks that the first call is ambiguous, and that is OK.

Tucker is in favor of hiding in this case.

After discussion, he changes his mind and says that it would be better to disallow the second operation (3). It would be illegal if the operation was inherited (an inherited operation cannot both be implemented and overridden; the example above is illegal if the declaration (A) is included); so it should be the same for other primitive operations even when not inherited.

Bob says that hiding is bad here, because you usually would want to call the outer one (which is not the one you would get). He and Gary both would prefer the ambiguous or error case.

Pascal does not like ambiguous. Randy doesn't like a preference rule to have hiding.

No one objects to making it illegal. Make it illegal. We vote to approve the correction: 7-0-4. [Editor's note: But we don't specify either wording or what we're correcting, so this vote is non-binding, and we'll look at it again in York.]

—

We review the Alan package AIs, which have been extensively reworked.

We look at C.7.3. C.7.3(5) should have a bold **return**. The indentation is weird.

Steve Baird asks whether the rules require the same access to handler to be returned that was set. That leads into a long and pointless discussion.

Alan insists on looking at his comments. D.1(20) needs updating, it is too specific to one policy; drop the second sentence and start the third with "Sources". Alan will fix the AARM note.

D.5.1(15). Replace the term "standard task dispatching policy" with "FIFO_Within_Priorities".

D.9(14) can be confusing, since we have a round-robin policy. Drop the note.

D.13.1(1) "names" should be "is".

D.14(13) Bob had asked why CPU_Time isn't defined at start. It should say that it is set to zero when the task is created. (It could get to be non-zero before the task is activated.)

D.14(5) Tucker worries that the function Clock is a weird name. He would think that it would be "Time_Used". Steve Michell and Randy agree. Pascal says overloading is good. But this isn't the same thing as Clock. Pascal does not want to discuss identifiers, it is too late.

D.14(25) had typo: there is an extra space in "execution-time".

D.14(24) should look like D.8(39).

D.14.1(7): "Canceled" should have mode **out**, not **in out**.

D.14.1(4): **not null access constant** discriminant.

D.14.1(15): The italics on “expired” should be moved to the first use. The last use “said to be” should be “said to have”. Alan also would add at the end of the third sentence: “; if in time is zero or less, the timer TM is said to have expired.”

Erhard says that we need to define “when a timer expires”. “In this mode, the timer TM expires when the execution time ....”. “If In_Time is zero or less, the TM expires immediately.”

Tucker has rewritten this while we’re discussing it:

The procedures Set_Handler associate the handler Handler with the timer TM; if Handler is null, the timer is cleared, otherwise it is set. The first procedure Set_Handler loads the timer TM with an interval specified by the Time_Span parameter. In this mode, the timer TM *expires* when the execution time of the task identified by TM.T.**all** has increased by In_Time; if In_Time is less than or equal to zero, the timer expires immediately. The second procedure Set_Handler loads the timer TM with the absolute value specified by At_Time. In this mode, the timer TM expires when the execution time of the task identified by TM.T.**all** reaches At_Time; if the value of At_Time has already been reached when Set_Handler is called, the timer expires immediately.

Pascal notes that TM.T should be TM.T.**all**.

D.14.1(26): Add at front: “As part of the finalization of an object of type Timer, the timer is cleared.” No, this should be in Dynamic Semantics, goes after paragraph 21.

D.14.1(11): Hyphen missing in “execution-time overruns”.

D.14.2(6): “<}” should be “<>“. (This is only in the AARM)

D.14.2(21): Change the second sentence to start with “The exception Group...”. Gary would prefer: “with To as the Time_Span value.” ending the first sentence.

D.15(3): The parameters aren’t aligned well. Probably should break this up with blank lines. But not to be inconsistent with other clauses of Annex D.

D.2.2(13.1): Alan thinks this is clunky. “An implementation shall allow locking policy (See D.3) Ceiling_Locking to be specified with one or more Priority_Specific_Dispatching for a single partition.” John and Pascal would prefer not to leave the “both”. Also in D.2.3(10), D.2.4(9), D.2.5(16).

Bob suggests “specifying” should be changed to “the specification of”. Alan and John like that; Steve Baird thinks it is a longer form of what it already says.

We have no consensus on a change. Alan will try to find a consensus with Pascal, Randy, and John.