Minutes of the 33rd ARG Meeting

9-11 November 2007

Fairfax, Virginia, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder , Ed Schonberg, Tucker Taft, Bill Thomas (Friday only).

Observers: Greg Gicca (except Sunday), Stephen Michell (Friday morning only), Luke Wong (Friday morning only).

Meeting Summary

The meeting convened on 9 November 2007 at 09:15 hours and adjourned at 11:55 hours on 11 November 2007. The meeting was held in a conference room at the Hyatt Fair Lakes Hotel in Fairfax, Virigina, USA. The meeting covered the entire agenda to the surprise of the editor.

AI Summary

The following AIs were approved:

AI05-0003-1/02 Qualified expressions and "names" (8-0-1)
AI05-0034-1/03 Categorization of limited views (9-0-1)
AI05-0036-1/01 Number of characters to be output for Text_IO for enumerations (7-0-2)
AI05-0058-1/01 Abnormal completion of an extended return statement (9-0-0)
AI05-0064-1/01 Redundant finalization rule (7-0-2)
AI05-0070-1/01 Elaboration of interface types (9-0-0)
AI05-0072-1/01 Termination only signals 'Terminated when it is True (9-0-0)

The following AIs were approved with editorial changes:

AI05-0006-1/02 Nominal subtypes for all names (8-0-1)
AI05-0022-1/02 Container tampering should be checked for formal subprograms (9-0-1)
AI05-0026-1/02 Missing rules for Unchecked_Unions (8-0-2)
AI05-0027-1/03 Behavior of containers operations when passed finalized container objects (9-0-0)
AI05-0032-1/01 Extended return statements for class-wide functions (7-1-1)
AI05-0033-1/01 Rules for non-library level interrupt handlers (9-0-0)
AI05-0038-1/02 Minor errors in Text_IO (7-0-2)
AI05-0039-1/01 User-defined stream attributes cannot be dynamic (8-0-1)
AI05-0042-1/01 Overriding versus implemented-by (9-0-0)
AI05-0044-1/02 Equivalence and equality in containers (7-0-2)
AI05-0045-1/03 Termination of unactivated tasks (9-0-0)
AI05-0048-1/01 Redispatching is not expected in language-defined subprograms (9-0-0)
AI05-0062-1/01 Null exclusions and deferred constants (9-0-0)
AI05-0065-1/01 Remote access types should be defined as externally streamable (8-0-1)
AI05-0068-1/01 Inherited subprograms may be both abstract and requires overriding (9-0-0)
AI05-0069-1/01 Singleton container (9-0-0)
AI05-0073-1/02 Questions about functions returning abstract types (9-0-0)

The intention of the following AIs was approved but they require a rewrite:

AI05-0013-1/06 No_Nested_Finalization is difficult to enforce (9-0-1)
AI05-0023-1/03 'Read on records with variant parts (9-0-1)
AI05-0052-1/02 Coextensions and distributed overhead (8-0-1)

AI05-0053-1/02 Aliased views of unaliased objects (6-0-1)
AI05-0057-1/01 The class attribute of a constrained subtype (4-1-4)
AI05-0059-1/01 Limited derived types and build-in-place (9-0-0)
AI05-0067-1/01 More build-in-place issues (8-0-1)
AI05-0071-1/01 Class-wide operations for formal subprograms (8-0-1)

The following AIs were discussed and assigned to an editor:

AI05-0030-1/02 Requeue on synchronized interfaces
AI05-0041-1/03 Can a derived type be a partial view?
AI05-0047-1/03 Annoyances in the array packages
AI05-0050-1/02 Return permissions are not enough for build-in-place
AI05-0051-1/03 Accessibility of dispatching function calls (aka another Baird question)
AI05-0054-2/02 Variable views of constant objects
AI05-0060-1/00 The definition of Remote access types is too limiting
AI05-0061-1/01 Assume-the-worst rule needed for access-to-discriminated checks
AI05-0063-1/01 Access discriminants on derived formal types
AI05-0066-1/01 Temporary objects are required to live too long
AI05-0074-1/01 Limited view of generic instantiations
AI05-0075-1/01 More access discriminant checks needed

The following SIs were approved with editorial changes:

SI99-0004-1/04 Changes to Asis for changes to access types (9-0-1)
SI99-0012-1/05 Add support for null procedure declarations (9-0-1)
SI99-0025-1/02 ASIS 99 allows too much variability between implementations (8-0-2)

The intention of the following SIs was approved but they require a rewrite:

SI99-0022-1/04 Add boolean queries to ease use of trait_kinds (10-0-0)
SI99-0028-1/04 What does appropriate kinds mean? (8-0-2)

The following SIs were discussed and assigned to an editor for further work:

SI99-0024-1/07 Provide a semantic model in addition to existing syntactic model

Detailed Minutes

Meeting Minutes

John repeats the errors that he had previously sent the editor. In AI-19, the "example in !example" should be "example in !discussion". In the first paragraph of that AI, "separately" should be "separate".

The minutes were approved unanimously with these changes.

Date and Venue of the Next Meeting

Our next meeting will be somewhere in the Tampa Florida area with Greg Gicca hosting, on February 8-10, 2008. Steve Michell notes that he is trying to schedule a POSIX RG meeting immediately before it. The ARG should reserve time at the beginning of the meeting to discuss PRG issues. Greg is hoping to locate the meeting in one of the beach cities. We ask him to keep the room rates to $100-150 – we don't want to price our volunteers out of participation.

The following meeting is after Ada Europe in Venice, Italy. June 20-22, 2008. WG9 is in the morning of June 20th, we'd start in the afternoon. Erhard tells us to make our reservations early (or pay double!). The latest date for rooms at the conference venue is February 29th. See the Ada Europe web site (soon but not yet). Erhard will send the ARG list a reminder when it is possible to make the reservations.

We ask Tucker if he can actually come next year. He replies (after much looking in planners) that his trip starts on June 24th, so it is compatible with the meeting dates.

Schedule of ASIS

Joyce said that she asked at the SC22 meeting to change the status of the ASIS project. This effectively has "stopped the clock". Otherwise, we would have had to be finished by next fall; we now will get an extra 18 months. (Total of 30 months.) We should have a draft finished by that time, although it does not have to be final.

Someone wonders what has changed, given that we handled ASIS the same as other standards that we have done. Steve Michell says that the difference is that JTC1 is now enforcing deadlines, which wasn't true in the past.

Discussion of ASIS Standard draft

We look at 15.36 as representative of the changes made. The grammar of "Returns a list of elements that each have one of Element_Kinds:" is hated. "Returns a list of elements, each of which has one of the following Element_Kinds:" We decide to postpone this discussion until the discussion of SI99-0028-1, since there are additional cases to consider.

Randy notes a bug in function Extended_Return_Statements (after 18.21), there is "wach", which should be "each".

Greg promises that a draft will be available in early January, presuming that Randy gets him the draft minutes and current ASIS source in a timely fashion. Randy takes an action item to do that.

ACATS test objective review

Randy provided a list of test objectives tested by the proposed new tests for ACATS 3.0.

We have a meta discussion about what should be tested. We agree that tests of rarely used features that are actually used in practice should be tested. For instance, selected components whose prefix is a call to a build-in-place function, is an example of a rarely used feature that needs testing. John says that in objective 6-02: "used in" should be "applies to". Objective 6-01 should be changed the same way.

John asks about 7-08. He isn't sure if it is right. Others agree that it is correct.

As an aside, Randy notes that he intended further tests of this using controlled components, but wanted to see how implementers reacted to this one first. It is noted that this also could be tested with 'Access (use a <> component), and that there would be value to trying both in the ACATS.

John wonders if 10-21 and the following objectives are complete? Yes, 10-17 and others cover the illegal cases. Randy notes that in general, he's only issued tests where an existing or submitted test could reasonably be used to cover the objective, so it is not unusual for all cases not to be covered with tests.

John and Tucker have typos that they will send to Randy off-line.

Approve test objectives with changes: 8-0-2.

Randy is asked the planned release date for the new test suite. He replies that it is planned to be released before the end of the year.

Thanks

Unanimously:

- We express our appreciation to Ed Schonberg for taking over the leadership of our group;
- We give our appreciation for the hard work of our outgoing chairman, Pascal Leroy;
- We bestow our appreciation for the work of our long-suffering editor and minute-taker, Randy Brukardt; [The Rapporteur adds that his notes say that the attendees lavished endless praise on the ongoing efforts of our editor, without whose unstinting labors the maintenance of the language would be doomed.]

- We give our appreciation to SIGAda for the fine facilities for the meeting.

Old Action Items

Steve Baird did not do the ASIS research project. Randy Brukardt did not describe a massive extension to Ada.Directories (AI05-0049-1). Greg Gicca did not do the definitions section, nor did he do the renumbering (which is not planned for the immediate future). Bibb Latting did not do an update to AI05-0009-1. Pascal Leroy did not do any of the SIs that had been assigned to him; they should be reassigned. He also said that he could not find a solution for AI05-0054-1.

Bill Thomas did SI99-0022-1 very late — after the agenda was mailed out. It was posted on the website and e-mailed to the group on Wednesday morning.

New Action Items

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

Steve Baird:

- AI05-0013-1
- AI05-0053-1
- AI05-0057-1
- AI05-0075-1
- Create an SI (later assigned number SI99-0035-1) outlining problems and possible solutions in the ASIS.Data_Decomposition package (see discussion of Jean-Pierre's request to find Size during the June 2007 meeting).

John Barnes:

- AI05-0030-1
- AI05-0047-1

Randy Brukardt:

- AI05-0049-1
- SI99-0028-1 (with Greg Gicca)
- Provide Greg Gicca a draft of the ARG meeting minutes and the current ASIS standard source code as soon as possible after the meeting.
- Check that all SIs have an owner and are progressing, or have been approved or voted no action.
- Study whether it is possible to properly define "=" for types in ASIS rather than declaring "=" abstract - create an SI (later assigned number SI99-0033-1) to fix this if possible (see Details of the ASIS Revision in the Albuquerque minutes)

Editorial changes only:

- AI05-0006-1
- AI05-0022-1
- AI05-0026-1
- AI05-0027-1
- AI05-0032-1
- AI05-0033-1
- AI05-0038-1
- AI05-0039-1
- AI05-0042-1

- AI05-0044-1
- AI05-0045-1
- AI05-0048-1
- AI05-0062-1
- AI05-0065-1
- AI05-0068-1
- AI05-0069-1
- AI05-0073-1
- SI99-0004-1
- SI99-0012-1
- SI99-0025-1

Gary Dismukes:

- Ambiguity of prefixed views (see discussion below) - later assigned number AI05-0090-1

Bob Duff

- AI05-0050-1
- AI05-0067-1
- Containers should be Remote_Types (see discussion below) - later assigned number AI05-0084-1

Greg Gicca:

- Add newly ARG approved SIs to the draft ASIS standard.
- SI99-0028-1 (with Randy Brukardt; add to standard even though not approved)
- Create a definitions section for the draft ASIS standard and a matching SI (see ASIS standard format in the Paris minutes) - later assigned SI99-0030-1
- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format in the Paris minutes)

Bibb Latting:

- AI05-0009-1

Brad Moore

- AI05-0060-1

Erhard Ploedereder:

- AI05-0054-1

Ed Schonberg:

- AI05-0052-1
- AI05-0059-1
- AI05-0063-1

Tucker Taft:

- AI05-0023-1
- AI05-0041-1
- AI05-0051-1
- AI05-0061-1
- AI05-0066-1

- AI05-0071-1
- AI05-0074-1 (create an alternative describing rules for **end private**)
- SI99-0007-1 (possibly add to SI99-0024-1)
- SI99-0019-1
- SI99-0021-1 (possibly add to SI99-0024-1)
- SI99-0024-1

Bill Thomas:

- SI99-0022-1
- SI99-0023-1
- SI99-0029-1

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment AIs. The AIs and SIs 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.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of ASIS Issues

SI99-0004-1/04 Changes to Asis for changes to access types

In paragraph 2 of the Summary, "existance" should be "existence".

In the question "defition" should be "definition". In the wording for Is_Task_Definition_Present, "defintion" instead of "definition".

Approve SI with changes: 9-0-1.

SI99-0012-1/05 Add support for null procedure declarations

The summary should mention abstract subprograms.

Spelling errors: "proceduree" in discussion. "accomodate" in discussion should be "accommodate". In Is_Abstract_Subprogram wording, "subrprograms".

Bill wonders if these routines are going to be confused with the similar semantic domain routines. These are syntax routines (for instance, renaming of them is not covered, as they are not declared). Add a user note that calling these on a renames of an abstract routine or null procedure returns False (these are syntactic), possibly suggest using the semantic one instead if that is important. The editor should invent these notes.

Approve SI with changes: 9-0-1.

SI99-0022-1/04 Add boolean queries to ease use of trait_kinds

Typos: The subject is still wrong: "Boolean" should be capitalized. It ought to say "Add Boolean queries to replace Trait_Kinds", to reflect what we actually are doing.

In the first sentence of wording, "classication". In the 2nd paragraph of discussion, "incorrrect". In the 3rd paragraph, "addiing".

Bill left out Has_Tagged since it isn't clear from the minutes if it was supposed to be included. Tucker says that once you have these, you probably will be surprised if they cannot be handled this way.

What about other reserved words? **all**, **constant**, and **tagged** should be included. Gary asks about **until**, but that seems to be covered in another way. **overriding** would be confusing (it is included in **not overriding**).

"Need to update 3.9.4" is not acceptable wording. Bill wonders if this should just be deleted. Tucker says that this isn't adding an additional level of classification, so the existing traits should just be deleted from 3.9.4. Bill should also look though the standard to see if there are other uses of Trait_Kind that ought to be deleted or updated.

3.9.4 of the draft standard has a duplicate line talking about Expression_Kinds, that should be deleted. Probably a cut and paste error.

The wording of these should say "immediately". That seems unnecessary for most of these. But it is needed for "**all**", because an expression could have an "**all**" buried within it, and many declarations and statements include expressions in one form or another. Has_All seems to be more confusing than it is worth (does it include nested expreesions or not?), and we don't need it for any purpose, so it is best to not have it at all.

constant is messy, because nested anonymous access parameters can contain **constant**. So forget that, too. So we just need to add Has_Tagged.

Add something to the !discussion to the effect that we considered Has_All and Has_Constant, but rejected those routines for the reasons noted above.

Put these routines in alphabetical order.

Approve intent of SI: 10-0-0.

SI99-0024-1/07 Provide a semantic model in addition to existing syntactic model

Should this be compilable by an Ada 95 compiler? It's not critical, but it will help partial implementations of Ada 2005. Tucker prefers this Ada 95 structure anyway. But he shouldn't try to avoid new Ada features like **not null** and anonymous access-to-subprograms if they will make the interface more usable.

Steve Baird notes that there is more than one possible subprogram being overridden, so there has to be the possibility of returning a list for Overridden_Declaration.

He has functions that depend on predicates indented in these specifications.

Has_Stub is confusing. It means that this program unit has a stub; it does not mean that the unit contains a stub (which is what the name seems to imply).

Steve Baird wonders if there would be value in knowing if the completion is in the same compilation unit. Maybe.

The function Corresponding_Declared_View has a non-existent type, it needs be changed.

This interface definitely should be constructed using tagged types. Implementing inheritance structures with untagged types would look like a 1980s design.

Tucker will try to get Sergey and Jean-Pierre involved in creating some examples. Bill notes that there are examples in the ASIS annex that potentially could be used for ideas.

Tucker also should start on wording for these routines.

Steve Baird asks what to do if a declaration defines multiple views. `A, B : Integer;` The connection between the syntax and semantics is broken in this case, as there are multiple views associated with a single (syntactic) declaration. Tucker will try to find a fix, hopefully one that will not make the interface harder to use in the common case.

Keep SI99-0024-1 alive: 10-0-0.

SI99-0025-1/02 ASIS 99 allows too much variability between implementations

"anomolies" should be "anomalies" in the question.

Drop "with defined values" from the Recommendation.

"implict" in 7.13 of the Recommendation.

Rounded quotes around option in 2nd paragraph of the discussion; also apostrophes. Third paragraph has a emdash, use "--" instead. These characters show up bizarrely in some people's viewers.

In F.X.2 of the wording, the "not recommended" sentence has the wrong name for the function.

Tucker would like to replace "This query returns a constant value." by "This query always returns True" (or False), and remove that from the later paragraph in all of the wording. That does seem better.

In F.X.1, Tucker hates "also". Replace with, "In addition to the interfaces defined in section 7, ..." (and drop "also").

Approve SI with changes: 8-0-2.

SI99-0028-1/04 What does appropriate kinds mean?

Put "appropriate kinds" in quotes in the subject.

Tucker suggests "<param> expects an element of one of the following <something>_Kinds:" and "that is one of the following <something>Kinds:"

Randy says elements have a kind, they aren't "is a kind". Ed agrees, and eventually Tucker does too.

Tucker suggests rewording to "<param> expects an element that has one of the following <something>_Kinds:" and "that has one of the following <something>Kinds:"

That is, use "has" and "have" consistently, don't use "of" or "with".

"Returns an element that has one of the following <something>_Kinds:"

"<param_name> expects an element that has any <something>_Kinds."

But most of these say nothing at all (anything is allowed), and it would be better to simply say "<param_name> expects any kind of element." for those.

That doesn't work if we want to support only legal declarations: Declaration_Kinds includes Not_A_Declaration, so it’s the universal set! In such cases (where there is an exclusion), we should use the wording:...that has any <something>_Kinds except Not_A_<something>".

"<param> expects a list of elements that each have one of the following <something>_Kinds:"

And similarly for the other cases.

Approve intent: 8-0-2.

We'll make these changes directly into the standard, even though the SI isn't approved (otherwise the Standard would not match the SIs, given that it already was changed once). Randy and Greg will make a list any changes that they find that look unusual (read weird, etc.)

(Re)assignment of open SIs

A number of SIs were previously assigned and have not been progressing. We should reassign these SIs.

Tucker will take the action items for SI99-0007-1 and SI99-0021-1 to ensure that these are covered by the semantic interface. If appropriate, he can merge these SIs directly into SI99-0024-1.

Tucker will also take SI-0019-1.

Bill will take SI-0029-1. He thinks that the first list is the correct answer, the second list is a left-over from a permission in ASIS 83 that was removed in ASIS 99.

For SI-0023-1, we just need wording. Bill will take this.

Randy will check why "=" is just not defined to work properly.

Randy should check that all SIs have an owner and are progressing (or have been approved or killed). [Editor's note: All open SIs as of the end of the meeting are assigned to someone.]

Detailed Review of Ada 2005 regular AIs

AI05-0013-1/06 No_Nested_Finalization is difficult to enforce

Only the wording of the restrictions has changed. However, Erhard and Ed don't understand the original rule change. After much discussion, they understand why 7.6(9.4) was deleted, it doesn't have to be illegal.

Steve Baird wonders if T'Class is covered, as it always possible for it to have extensions that are controlled (or task or protected). No, but it obviously should be.

So we need to add a rule that a class-wide type needs finalization to 7.6. Add a bullet: "it is a class-wide type; or" (this replaces the deleted 7.6(9.4/2).

The restriction needs to be reworded, too, so that we don't disallow access T'Class. "There are no allocators where the type determined by the subtype_mark of the subtype_indication or qualified_expression needs finalization where the type of the allocator does not have library-level accessibility."

There is much debate about the meaning of restrictions. Tucker and Bob argue that these are *just* definitions and don't (directly) represent legality rules. So none of them should be "shall". Randy grumbles that many of these don't make sense; for example, D.7(3/2) makes no sense; it's not at all clear to whom this applies or how it is checked. Tucker claims that it is all explained in 13.12.

The original authors of this text carry the day. Remove "shall" from D.7(4, 10.1, 10.2, 10.8) as these are definitions of restrictions. Otherwise, leave these as they originally were. And phrase restrictions in terms of the excluded constructs as much as possible.

Steve Baird will look at whether the restrictions are checkable at compile-time (specifically No_Task_Hierarchy).

Approve intent of AI: 9-0-1.

AI05-0022-1/02 Container tampering should be checked for formal subprograms

Change the question from "finalize L" to "delete an element from L", as that would be a much more likely occurrence and makes it clearer why we need to fix this. Delete "Not Good." Change the last sentence to "we don't intend for erroneousness to be allowed".

Approve AI with changes: 9-0-1.

AI05-0023-1/03 'Read on records with variant parts

We try to remember what we had previously decided.

Remove "then" from the sentence that starts "Finally" in 13.13.2(9). "... creates an {anonymous} object...". "... for the created {anonymous} object."

We won't add "anonymous" to 13.13.2(27/2), that's ancient wording and it doesn't add anything.

Get rid of the parenthesized part in the text added to 13.13.2(27/2). The interaction with the permissions is nasty. Put this statement before the last sentence of 13.13.2(27/2).

Tucker wonders why we need this statement. Steve Baird says that we need to know what happens if Read tries to modify the discriminants of the object passed to it. (Imagine that Read is user-defined and tries to change the entire parameter.)

Tucker would prefer to turn this around. " If T has discriminants, then this object is unconstrained if and only if T has discriminants with defaults."

In the permission, "...create an {anonymous} object...".

Tucker suggests: "If T is a discriminated type and its discriminants have defaults then the default implementation of S'Read may not create an anonymous object of type T if Item is a constrained variable or its discriminants match those in the stream." Put the rest of the stuff into an AARM note (what happens if the object is eliminated).

Erhard objects because he wants it to be explicit that the Constraint_Error is raised. The AARM note would cover that; there is no permission to remove the Constraint_Error, *only* the anonymous object.

Randy says that we don't need the second permission at all, because S'Input can arrange the discriminants to have the correct values, and then the first permission would apply. After some thought, Tucker and Steve agree. But there is a bit of weirdness: a user-defined Initialize routine could be surprised by an unconstrained object with non-default discriminants. Is this a real issue? We decide to fix it by allowing a constrained object:

"The default implementation of S'Input that calls the default implementation of S'Read may create a constrained anonymous object with discriminants that match those in the stream."

Tucker will take this AI. He notes that the wording of 13.13.2(27/2) is wrong, as it implies that the discriminants are read from the stream for types with discriminants that have defaults.

Approve intent of AI: 9-0-1.

AI05-0026-1/02 Missing rules for Unchecked_Unions

Remove question 3 (this was supposed to have been done already, as it was in the minutes from last time).

The wording for 7.6(9.4) should say "partial view" as opposed to "private type".

"I" in question 2 should be changed. "There appears to be no rule that has that effect.". Delete the part about the minutes.

Approve AI with changes: 8-0-2.

AI05-0027-1/03 Behavior of containers operations when passed finalized container objects

!wording is missing before the wording.

In the example, Finalzation should be Finalization.

Perhaps we should add a To Be Honest AARM note that this requirement only applies to items declared in the visible part. No, we should just add "in the visible part of" to the wording. "It is a bounded error to call any subprogram declared in {the visible part of} Containers.Vectors..."

Last word of the discussion should be cursors.

Erhard asks that the first sentence of the summary read "It is a bounded error if ..."

Approve AI with changes: 9-0-0.

AI05-0033-1/01 Rules for non-library level interrupt handlers

In the summary, "nested" should be "declared".

In part (2) of question, "seems like [is] {it}..."

In the wording for part (1), "these rules apply ..."

Erhard wonders why the 13.11.2(16) rule is needed. Randy explains that there is no object here (this is just an access-to-subprogram), and Unchecked_Deallocation could have been done on the protected object after the point that 'Access was created and before the use.

We have a long discussion of about erroneous behavior of renames of prefixed views which seems similar to this case, but is not because a name of the non-existent object will be directly referenced by the called subprogram (by evaluating the name of the formal), and that triggers the existing rule. That doesn't happen for protected subprograms, where the name of the object is implicit (thus no evaluation of a name to trigger the existing rule).

13.11.2(16). Merge the two rules. "Evaluating a name that denotes a nonexistent object or a protected subprogram whose associated object is nonexistent is erroneous."

Approve AI with changes: 9-0-0.

AI05-0034-1/03 Categorization of limited views

Approve AI: 9-0-1

AI05-0036-1/01 Number of characters to be output for Text_IO for enumerations

Approve AI: 7-0-2.

AI05-0038-1/02 Minor errors in Text_IO

Spelling error: Verbatuim in editor's note.

In question (2), "over" should be "cover".

"If the value specified by To is less than the current line number, has the effect of calling New_Page followed, if To is greater than 1, by a call to New_Line with a spacing equal to (To - 1).

Why the change to A.10.8(8)? Bob said (in e-mail) that it should be the same as Integer. Bob (here) isn't sure where that came from. We don't want to make an incompatible change for no defined reason. OTOH, the different behavior for Integer and Modular seems illogical and not helpful to users.

A.10.8(8) is also inconsistent with the note A.10.8(24). That note is broken: 0..Num'Last doesn't match either the current or proposed wording. After some quick checking, we find that GNAT and Sofcheck's compilers both operate following the proposed wording. [Editor's note: So does Janus/Ada and IBM Rational Ada. Also, the original wording is inconsistent with the way Get from a String works for modular types.] We'll keep the change.

Delete the note A.10.8(24), fixing it would not provide any useful information.

Approve AI with changes: 7-0-2.

AI05-0039-1/01 User-defined stream attributes cannot be dynamic

There are various typos. In the question, Subrprogram is misspelled. In the fourth paragraph of the questions, "its" should be "the". In the first line of the question, AARN should be AARM.

Approve AI with changes: 8-0-1.

AI05-0041-1/03 Can a derived type be a partial view?

Bob asks that "which" be changed to "that" in the wording for 3.7.1(7/2), 4.8(6/2), and in the 2nd paragraph of the discussion.

Tucker says that the wording of 3.3(23.10/3) is confusing. Randy suggests using more bullets:

For the purposes of determining within a generic body whether an object is known to be constrained:

- if a subtype is a descendant of an untagged generic formal derived type, it is not considered indefinite and is considered to have a constrained partial view;
- if a subtype is a descendant of a formal access type, it is not considered pool-specific.

In 3.3(23.9/3) and 3.7.1(7/2) replace with "constrained partial view" to consistently use the terminology. Also in 4.8(6/2) "that has a constrained partial view of the designated type" and associated AARM notes.

Actually, 4.8(6/2) should be rearranged....

"there is a constrained partial view of an ancestor of the designated type;".

"there is an ancestor of the designated type that has a constrained partial view;"

Tucker and Bob worry that this will impact tagged types that would not be affected by the original problem (as a tagged type could never have mutable discriminants). Bob thinks these rules should not ever apply to tagged types. The Amendment has no such exception. What about untagged partial views of a tagged type? Ugh, this is not a trivial change.

Tucker will take this and try to figure out what the rules ought to be, as opposed the ones he proposed (and we adopted) for the Amendment.

AI05-0042-1/01 Overriding versus implemented-by

Add a question to the question. "Should this example be legal? (Yes.)"

Why did we not call this overriding? Erhard thinks that it was because the profiles of the operations are different, and we didn't want to mess with the overriding rules for this case. That seems like a likely explanation.

This seems to be a lot of wording that doesn't say much; users may be confused by the "the effect of" wording. Tucker suggests just adding the new wording to the end of the existing paragraph:

If the first parameter of a primitive inherited subprogram is of the task type or an access parameter designating the task type, and there is an entry_declaration for a single entry with the same identifier within the task declaration, whose profile is type conformant with the prefixed view profile of the inherited subprogram, the inherited subprogram is said to be *implemented* by the conforming task entry {using an implicitly declared non-abstract subprogram which has the same profile as the inherited subprogram and overrides it}.

Use similar wording for the protected case.

Bob would like an AARM note and index entry in 8.3 that points to these two places.

In unrelated news, Tucker says that "needs finalization" needs more index entries. Specifically, there should be an index entry for each standard type defined to "need finalization". This should be added to AI05-0005, along with many other non-normative changes.

There is a typo in the summary: "implict" should be "implicit".

Regarding the second paragraph of the question, first line, Gary asks about "there appear to be a few places". This leads to a long pointless argument about this grammar. No change is required.

Approve AI with changes: 9-0-0.

AI05-0044-1/02 Equivalence and equality in containers

Tucker notes a typo "is be" in the first sentence of the wording. Erhard notes "strict weaking order" instead of "strict weak order" in the same sentence. Ed finds " an unwarrented assumption." in the question. Erhard finds that the discussion has "What is not OK [if]{is} for there to be...". Randy notes "...which [has] is equivalent..." in the same paragraph.

"An operator "<" defines a "strict weak ordering" if it is irreflexive, asymmetric, transitive, and in addition, if x < y for any values x and y, then for all other values z, (x < z) or (z < y)."

No one can figure why we need to define "smallest first"; the existing text already says that the formal "<" determines the order. So drop this wording. We continue to discuss this for a long, long time.

Erhard worries that "<" could be ">". Sure, but no one will be confused by the current definition, and the attempt to define it does not add any clarity (if anything, it makes it less clear).

A "strict weak ordering" is stronger than what we called a "strict ordering". Another reference called that a "strict partial ordering", which would be less confusing. But we're going to get rid of all uses of "strict ordering" here, so that doesn't matter.

Erhard notes that we need a lead-in sentence for the definition of A.18.2. Otherwise, it appears out of the blue. "When a formal operator "<" is used to provide an ordering for a container, it is generally required to define a strict weak ordering. An operator..."

A.18.8(65.2) should be A.18.8(65/2). This wording should be a separate paragraph, after A.18.8(66/2) and A.18.9(79/2).

Approve AI with changes: 8-0-2.

AI05-0045-1/03 Termination of unactivated tasks

Change the wording: "...as {a} part of the return object or one of its coextensions immediately becomes terminated and is never activated [regardless of where it would have been activated].

This is still not liked much. It would be better to put the common case first.

If the master that directly encloses the point where the activation of a task T would be initiated, completes before the activation of T is initiated, T becomes terminated and is never activated. Furthermore, if a return statement is left such that the return object is not returned to the caller, any task that was created as a part of the return object or one of its coextensions immediately becomes terminated and is never activated.

Reverse the notes in the same way. Bob notes that ATC is also abort, so the note should just say "abort", not "task abort". Steve Baird would like to drop the wording after "point".

Delete the Editor's note because we deleted the text in the brackets.

Approve AI with changes: 9-0-0.

AI05-0047-1/03 Annoyances in the array packages

This AI isn't written up.

For question 1, Ed suggests that the accuracy requirements for multiplication of matrices be removed completely; we shouldn't second guess the people that have designed BLAS. John agrees that we should just drop this requirement.

For question 2, use the same range for the result.

For question 3, it is not clear what to do - what does BLAS do? Research is needed.

For question 4, should these be Natural? It was suggested that you might want to have a symmetric range around 0 (-3..3). So leave this alone.

Question 5 is just an obvious error, it should be fixed.

Question 6: We should leave it and put in an AARM note.

Question 7: It is not clear what to do. We should research what BLAS does, and if they specify anything, then we should do so. Steve B. notes "...it amount{s} to..." in this question.

John will write up this AI.

AI05-0048-1/01 Redispatching is not expected in language-defined subprograms

There are double spaces between words in the question.

Remove the "unless" from the wording, as a overridden user-defined subprogram is neither inherited nor language-defined.

Tucker would prefer to replace "nonoverridden" with an "inherited". "behavior" should be "effect". "affect the effect"? Yuck. Use "alter" instead.

"In particular, overriding a language-defined subprogram shall not alter the effect of any inherited language-defined subprogram."

The AARM note has "doesn't not", drop the "not".

"For a descendant of a language-defined tagged type, the implementation shall ensure that each inherited language-defined subprogram behaves as described in this International Standard. In particular, overriding a primitive subprogram shall not alter the effect of any other inherited language-defined subprogram, unless otherwise specified by this International Standard."

Perhaps we should be more specific that we're talking about something the user does.

"If a descendant of a language-defined tagged type is declared, the implementation shall ensure that each inherited language-defined subprogram behaves as described in this International Standard. In particular, overriding a primitive subprogram shall not alter the effect of any inherited language-defined subprogram, unless otherwise specified by this International Standard."

The Discussion has "standrd" instead of "standard".

The AARM note should say that redispatching is not permitted unless it is required by the standard.

Approve AI with changes: 9-0-0.

AI05-0050-1/02 Return permissions are not enough for build-in-place

Tucker says that there is far too much wording proposed here.

Steve Baird says that we can't allow Constraint_Error to be raised at any random point in the execution. Thus he thinks we need this fairly complex permission. Tucker seems to think that it is OK to raise the exception anywhere. Randy complains, you don't want to allow this check to be triggered in nested subprogram calls, or for that matter in any other handler other than the ones specified by Steve.

Bob Duff will rewrite this wording. Tucker would like to see Bob's version before the next meeting.

AI05-0051-1/03 Accessibility of dispatching function calls (aka another Baird question)

Tucker had sent this version, but he has effectively withdrawn this idea in later e-mail discussions. He tries to explain his new idea. Essentially, it may require additional checks at the call site.

Steve tries to describe reasons why this might not work.

Tucker wants to use a mechanism similar to build-in-place to cover anonymous access return.

Randy wonders what is currently done. The current rules essentially require a storage leak because the pool doesn't allow deallocation and most functions are declared at library level. Tucker wants to fix that, but an equally good solution is to simply disallow such things by declaring the storage size of such types to be zero (then passing around pools is not needed, as no allocators can be written).

Tucker will write up his new idea for future battles.

AI05-0052-1/02 Coextensions and distributed overhead

The recommendation has "a coextensions",

Ed wonders if we are losing too much by making this illegal. No one seems to think so; this capability was added by the Amendment so there cannot be a significant compatibility problem. Do we agree on static checks for this? Yes; no one objects to that.

Steve would like a term for types that cannot become non-limited. They once were called "inherently limited"; perhaps we could use that.

Tucker says that non-formal limited private tagged types should be included as "inherently limited" (the full type of such a private type has to be limited).

"task or protected type, or for a type that is a descendant of an explicitly limited record type" would be OK.

What about generic formal types?

Ed will take this one to check/update the rules for generics and the rules for tagged non-formal types, and to possibly define a term for this rather than repeating the rule in a number of places.

Approve intent of AI: 8-0-1.

AI05-0053-1/02 Aliased views of unaliased objects

This was previously decided, we are just deleting the syntax.

Change the summary to "Return objects cannot be aliased."

Change the recommendation to (See summary.)

Change the question's answer to "Disallow aliased return objects".

Tucker wonders if we should allow this for tagged types. Bob says that he uses this to make self-references. Randy complains that it would be really weird to allow the keyword but only in a few cases.

But we could make some of these objects implicitly aliased, like tagged parameters. We would want to be able to do whatever you could do within a current instance of an explicitly limited type. The rules probably should be very similar to the current instance rules.

Steve will add this extra stuff to the AI.

Approve intent of AI: 6-0-1.

AI05-0054-2/02 Variable views of constant objects

Ed wonders if a static check is possible. No, because you can make these variable views in user-defined Initialize routines. Moreover, any check would have to be conservative in generic bodies, and that would disallow just about all constants there.

Change "is" to "denotes" in the wording: "A name that statically denotes a constant object denotes a constant view. A name that statically denotes a variable object denotes a variable view."

Erhard worries that high integrity users will not like this; he claims that he prefers erroneousness in this case. How safety-critical people would prefer erroneousness to defined behavior is not clear to this writer; others agree. Tucker notes that variable views can only be created via current instance, Initialize, Finalize, and (via AI05-0053-1), extended return objects.

Bob would like to get rid of the term constant object, because it is misleading. But that does not seem very worthwhile. No one is going to stop referring to constant objects.

Erhard is still worried. This may need to be detected by an analyzer, it cannot be done by the language because so much would have to be disallowed.

Ed suggests that we add a user note of some sort saying that constant objects of elementary types only have constant views. After all, you can only get a variable view of a non-limited type via Initialize and Finalize. Thus, this can only happen for limited types and controlled parts.

Note that Gnat has a warning to add **constant** when possible.

Erhard wonders why is **constant** on these sorts of objects in the first place? Because the client does not know how the private type is implemented. Moreover, the client might add **constant** for whatever reason - the object may very well be constant from their perspective. There also is a potential maintenance hazard if someone changes the implementation of the private part, we don't want clients (that may not even be aware of the maintenance) that have declared constants to break from such changes (since an erroneous program can do anything at all).

Erhard wants to try to see if there is some way to restrict this issue to private types. He doesn't think that this should be allowed when the type is visible. Tucker warns that generic contract problems may lurk there.

Straw poll on alternative 2 essentially as written: 5-1-3.

Erhard takes an action item to try to find an alternative solution (to replace AI05-0054-1), or to admit defeat as Pascal has done.

AI05-0057-1/01 The class attribute of a constrained subtype

Steve Baird shows us his three alternatives.

We discuss implementation for a while.

The runtime check in the body of #3 is pretty weird, but would be done at compile-time in non-shared generic body. Steve says that the problem would require a check to occur on the evaluation of a subtype_mark, which we don't have for anything else.

Tucker adds a suggestion that check is made when you elaborate the generic body, if it contains any S'Class. Steve complains that this would happen even for a use of S'Class in something will never be executed. Steve suggests a post-compilation check for this, but of course that is harder to implement.

We take a straw poll on which choice is preferred by each ARG member: 1-4; 2-0; 3-1. A lot of people abstain.

Someone might have existing code that depends on the current rules, so #2 is not good. (Of course, it is just as likely that they depend on the #2 semantics, since that is what most compilers implement.)

Steve will write up a full AI based on the intent.

Approve intent of AI: 4-1-4. (Randy still objects; he cannot stand the idea of unconstrained types with constraints; he has objected to that since at least 1993).

AI05-0058-1/01 Abnormal completion of an extended return statement

Normal completion of a compound statement includes exiting with a goto. Ouch.

Approve AI: 9-0-0

AI05-0059-1/01 Limited derived types and build-in-place

The braces in the wording for 3.7(10/2) should be brackets. (It is a deletion.)

Perhaps it would be better to define "inherently limited", which would cover all of this and then we wouldn't need to make these changes.

Bob says AARM 3.7(10.a/2) would need to be updated for this formulation.

We should consider doing that as part of this AI — and then use the term in AI05-0052-1. Ed will take this AI, so he can do both at the same time.

Approve intent of AI: 9-0-0.

AI05-0060-1/00 The definition of Remote access types is too limiting

Brad will take this on.

AI05-0061-1/01 Assume-the-worst rule needed for access-to-discriminated checks

Fix the spelling in the subject.

In the discussion, "dyanmic semantics".

Tucker thinks this should be combined with AI-41, so he will take this and do that.

AI05-0062-1/01 Null exclusions and deferred constants

Change the summary to "A full constant may have a null exclusion even if the deferred constant does not."

Bob asks that the discussion add that this is a useful thing to do for a private type whose full type is an access type. In that case, the null exclusion would not be allowed on the private type.

Question: "asymetrical" should be "asymmetrical". "A compiler writer [was]{has}..."

Approve AI with changes: 9-0-0.

AI05-0063-1/01 Access discriminants on derived formal types

This is another "inherently limited" problem. So this should be handled with AI05-0052-1 and AI05-0059-1, as such, poor Ed will have to deal with this one too.

The generic boilerplate will be added if necessary, and generic formal limited private will be "inherently limited" in the spec (not the body) of a generic.

AI05-0064-1/01 Redundant finalization rule

Randy explains that the rule (which was needed in Ada 95) was made redundant by the additional masters we added in the Amendment. It also fails to cover many cases, so it is either misleading or wrong, and thus it is best to get rid of it.

Approve AI: 7-0-2

AI05-0065-1/01 Remote access types should be defined as externally streamable

The proposed E.2.3(14) is impenetrable, according to Tucker. John says that if you read it slowly, it is OK.

Bob still wants to define these as having external streaming. Randy complains that that does not match the intuitive meaning of external streaming - there is no reason to assume a remote access type will work when written to a file, for instance. Bob looks up the definition of external streaming, and notes that it only talks about sending values between partitions. It's probably better to add these to the definition of external streaming.

It is suggested to add "non-remote" into the wording about access types. Tucker suggests adding a parenthetical remark to the last sentence ("All other types [(including remote access types, see E.2.2)] support external streaming."); that will handle the needed forward reference (and to reinforce the intent). Also change the AARM note to say "non-remote".

Here's the entire paragraph 13.13.2(50/2) as modified:

[A type is said to *support external streaming* if Read and Write attributes are provided for sending values of such a type between active partitions, with Write marshalling the representation, and Read unmarshalling the representation.] A limited type supports external streaming only if it has available Read and Write attributes. A type with a part that is of a non-remote access type supports external streaming only if that access type or the type of some part that includes the access type component, has Read and Write attributes that have been specified via an attribute_definition_clause, and that attribute_definition_clause is visible. [An anonymous access type does not support external streaming.]All other types [(including remote access types, see E.2.2)] support external streaming.

The existing wording changes in the AI are dropped.

Delete the discussion in the AI, and replace it by a paragraph saying that we thought it was easier to define this as supporting externally streaming rather than adding more text to the annex E.

Approve AI with changes: 8-0-1.

AI05-0066-1/01 Temporary objects are required to live too long

Randy explains the problem: function results are required to live too long for object initializations.

The rule probably is right for build-in-place, as there is no temporary return object; thus we need the object to live as long as the declaration.

Tucker thinks that a local change to either 3.10.2(10) or 7.6.1(13) would fix this.

Bob wonders why we don't require build-in-place for all aggregates that initialize objects? Because it would make implementers do work for no specific reason other than to remove a few words from the Standard.

This is really only an issue for functions returning non-limited controlled parts. [Editor's note: Aggregates with non-limited controlled subcomponents should also be included as an issue, as 7.6(17.1/2) applies only to aggregates of controlled *types*, not to aggregates with controlled *parts*.]

Tucker will take this AI and attempt a solution.

AI05-0067-1/01 More build-in-place issues

Steve (and Pascal) would like to define the model more semantically. Randy notes that he doesn't see sufficient non-pathological problems to make it worth lots of revision to the Standard. Ed comments that question #1 seems to be pathological for any program that depends on knowing which tag was used.

Bob describes a possible model for describing this. Tucker finds that weird and describes his "poof" model of this.

Bob is willing to do this AI, but only if we give him direction. We need a definition of the model of build-in-place. Bob would like a bit more detail. Steve would like to be able to derive answers to questions like these from the proposed rules (and the existing ones).

Approve intent of the AI: 8-0-1. The intent is essentially that Bob do a good job updating the rules.

AI05-0068-1/01 Inherited subprograms may be both abstract and requires overriding

The answer to the question "Legal?" is (No.)

The ACATS test section is from some other AI.

Randy explains that the existing wording is not clear that these are view-specific. Bob would prefer that we just make an AARM note to make these cases clear. Tucker thinks that the wording change does not help.

So make this make this a ramification, and add an AARM note.

Approve AI with changes: 9-0-0.

AI05-0070-1/01 Elaboration of interface types

Approve AI: 9-0-0.

AI05-0072-1/01 Termination only signals 'Terminated when it is True

Tucker explains the problem. If 'Terminated is False, it doesn't tell anything about the state of the other task.

Approve AI: 9-0-0.

AI05-0073-1/02 Questions about functions returning abstract types

In question (1), add two missing words: "should the generic {function} be illegal?" and "Also, note that this similar {generic} package:"

There is an error in the wording addition for 3.9.3(8/2): "designated" should be "designating".

There also is an error in the wording change inserted after 6.5(8/2) "...designating a specific {tagged} [access] type...".

John notes an extra word in the Discussion: "...was [be] made..."

We discuss if the return check should be limited to primitive operations. That looks messy, because primitive functions can be renamed as non-primitives (and vice-versa). So some sort of wrapper would be required for the check, and that is too much trouble and complication.

Approve AI with changes: 9-0-0.

AI05-0075-1/01 More access discriminant checks needed

Some of this wording change really belongs in AI05-0032-1; it has nothing to do with this problem. Specifically, the first bullet of 6.5(5.6/2) belongs in AI05-0032-1.

The added checks are two pairs of static and dynamic checks (as usual for accessibility checks).

Steve Baird thinks that "noninherited" is needed to prevent the static rejection of things that would not fail a dynamic check. He thinks that any inherited discriminants must be constrained. Randy thinks that you can inherit the old discriminants if you don't give any new ones, and surely the checks need to apply in that case. Tucker agrees; you can inherit them without a constraint. If the new type doesn't have a discriminant part, they are just inherited.

Steve says he was trying to avoid worrying about discriminants that were not supplied at the point of the extension. No such luck. Drop "noninherited".

4.8(5.2/2) "This condition shall also hold..." is bizarre; we don't use wording like that in the Standard.

Tucker suggests combining this rule into 4.8(5.2/2) "If the type determined by the subtype_indication or the qualified_expression of the allocator has one or more access discriminants, then the accessibility level of the anonymous access type of each access discriminant, as determined by the subtype_indication or qualified_expression of the allocator, shall not be statically deeper than that of the type of the allocator (see 3.10.2)."

This works because the designated type is the same as the specified type, unless the designated type is class-wide, so we cover Steve's wording.

Ed notes that this wordsmithing is not appearing productive for the whole group. Steve ought to redo the rest of the wording in the same way as suggested above. Ed asks that Steve provide examples of each of these rules failing. Randy is happy about that as it will provide a ready-made ACATS test.

Steve will update the AI to improve the wording and provide some examples.

Ambiguity of prefixed views (new topic)

Gary is worried about an ambiguity with prefixed view of an operation of a synchronized interface and the direct call to the entry. Tucker seems to think this is an existing hole in the language which is not related to any existing problem. In particular it is not related to AI05-0042-1.

4.1.3(9.2/2) does not allow prefixed notation if there is a component. Perhaps that needs to be extended to cover entries and (protected) subprograms. The rule has to exclude homographs of the prefixed view.

Gary asked this question, so he gets to write an AI on this topic.

Containers should be Remote_Types (new topic)

Bob wants an new AI on Remote_Types for containers. Pascal and Randy had objected to this (as putting heavy restrictions on implementations that don't even support Annex E) and intended to bury it. Tucker does not like that; Randy reminds him of the procedure for making an AI over the objections of the "management", and Tucker seconds Bob's suggestion. This action item will be assigned to Bob, and the associated mail will be moved to the new AI from AI05-0060-1. Bob should take into account the objections voiced in e-mail by Pascal.

Detailed Review of Ada 2005 amendment AIs

AI05-0003-1/02 Qualified expressions and "names"

The change is to allow qualified expressions to convert a name or expression into a constant view; that can then be used as a name in any context requiring a name.

Tucker says this actually does not provide any new functionality. He notes that a type conversion of any qualified expression is already a name, so there is little reason to disallow a qualified expression alone from being a name as a well. (This workaround comes as a surprise to everyone; we wonder how many compilers will fall over when given that in an unusual context??) That applies even if the item being qualified is a literal or aggregate.

Bob wonders if we should go farther. He wonders if parenthesized expression should be included. We'd also need (bare) aggregates in that case, and that makes some people uncomfortable.

Randy points out that the language never defines any semantics for parenthesized expressions in the normal case (there are a few special cases scattered throughout the standard). It doesn't even say that the value and properties are preserved by parens. So it can be anything we want. :-) People are more, rather than less, uncomfortable after that explanation.

Tucker did look through the whole standard for qualified expression, and 5.4 is the only usage that he felt that needed a change.

Ultimately, we agree that Tucker has this exactly right as written.

What happens to this AI? It just waits until WG 9 asks for Amendments. It doesn't seem important enough, especially given there is a workaround, for it to be a Binding Interpretation.

Approve AI: 8-0-1.

AI05-0006-1/02 Nominal subtypes for all names

Tucker says that this should not be classified as an Amendment, as it fixes a hole in the language. It should be a Binding Interpretation. Convert the AI to the format of a Binding Interpretation.

Erhard asks that we move the "unless":

"Similarly, unless explicitly specified otherwise, for an attribute_reference that denotes a function, when its result type is scalar, its result subtype is the base subtype of the type, and when non-scalar, the result subtype is the first subtype of the type."

Fix the AARM note as noted by Bob Duff in the appendix.

Approve AI with changes: 8-0-1.

AI05-0030-1/02 Requeue on synchronized interfaces

Why is this a bounded error? Shouldn't we just raise Program_Error here?

Why did we decide that last time? There is no value to having it incompatible between implementations; Bounded Errors are best for checks that are expensive, and this is not expensive. Just make this a Program_Error.

Erhard wonders why we don't have entries on interfaces? We should have done that, but it is rather late for that.

Ed worries that there no way for the user to see in the interface whether it is OK to requeue. He wants only a compile-time check.

One option would be to define a pragma to specify that something is implemented by an entry. Erhard suggests that we also have a pragma that requires implementing by a procedure (not an entry) [because those could block]

Bob notes that that is poor taste for a pragma. Randy notes that we've previously preferred syntax for such things. The rest of the group doesn't seem swayed by either argument; we're looking at providing this missing functionality sooner rather than later, and a pragma is a smaller change.

Straw poll: Should there be a pragma to check this at compile-time: 8-0-1.

John will take this and write a proposal.

The most important pragma is "implemented (only) by an entry"; the second case for a pragma is "implemented (only) by a protected procedure" (which implies no blocking). The second pragma can apply only a protected interface.

AI05-0032-1/01 Extended return statements for class-wide functions

We would need to add the rule that was mistakenly in AI05-0075-1:

Append after 6.5(5.3/2) (as a new item in the same bulleted list):

- If the result subtype of the function is class-wide, the accessibility level of the type of the subtype defined by the return_subtype_indication shall not be statically deeper than that of the master that elaborated the function body.

Steve wonders if this would allow anything that was not previously allowed. Tucker says no, it should be the same as a class-wide return object initialized by a function.

Should this be a binding interpretation? A straw poll comes out 7-1-1. Randy objects that we are adding too many holes to the pure static matching model for extended returns. (He adds that this feeling may be somewhat irrational.)

Erhard asks if there is an interaction with AI-57 (S'Class). Yes there is and Steve will add that to his AI.

In !proposal, "where Func is a dispatching call." should be "where Func returns a specific type." since the initializing function is not required to be dispatching. That's still confusing, replace both the leading and trailing words. "Instead, you have to write something like:" "where the result type of Func is the desired specific tagged type".

Bob wonders if the redundant clause that ends the addition to 6.5(5.8/2) is helpful; he thinks it is not valuable. Several people agree, as it is unlikely and surprising for such a conversion to raise an exception. (It can happen, however.) Remove that part.

Randy thinks that the change to 6.5(23/2) is redundant. Tucker says that the return object has a specific type, and the caller needs to see a different view. Randy thinks that will cause problems in the rest of the language; we have no provision for different views of the return object. Bob thinks he will have to take that into account for build-in-place.

Approve AI with changes: 7-1-1 (Randy objects because he thinks that the return object having the wrong type is a problem that requires more research).

AI05-0069-1/01 Singleton container

Procedure Clear should have an **in out** parameter.

John complains that the subject should be "Holder container"; there is no sense in using a different term in the Subject and the package name.

Randy will have to supply the missing wording.

Approve AI with changes: 9-0-0.

AI05-0071-1/01 Class-wide operations for formal subprograms

There is no equal operator for class-wide types for generic formal parameters. But if you just used "=" in the body of a generic (when the actual type was class-wide), that would in fact work. That is weird - you can pass it implicitly but not explicitly. The discussion of the AI should mention this fact.

Tucker suggests adding a bullet after 12.5.1(23/3). It tries to share the wording of previous bullets. Steve objects that this is static semantics; whereas these bullets are dynamic semantics.

Ed wonders about the implementation overhead. Steve thinks that the overhead comes from the Amendment rules, and this will not add much.

Tucker will write up the wording, and split it into static and dynamic semantics. This should be changed to a Binding Interpretation (because of the fact that "=" will work inside the package, so there is just some sort of omission). Tucker will need to reorganize the AI into the format of a Binding Interpretation.

Approve intent of AI: 8-0-1.

AI05-0074-1/01 Limited view of generic instantiations

Tucker would like a complete write-up of **end private** before we discuss this one. He now has an action item to do that.

Steve and Tucker talk about various solutions, many proposed in the past (such as the Atlanta solution). Ed repeats that changes to the freezing rules are not acceptable to him. It is suggested that Steve should have the same opinion (because Pascal held that opinion, and Steve now has to maintain his code). Steve laughingly agrees that perhaps he ought to study that part of his compiler more before proposing solutions. We realize that lunch is more important than continuing to hash on this topic. [Editor's note: this was the last topic discussed at the meeting.]