Minutes of ARG Meeting 63L

11 December 2025

Attendees:

Voting Members:

Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Christoph Grein, Jean-Pierre Rosen [arrives at 11:05 EST], Justin Squirek, Tucker Taft, Tullio Vardanega, Richard Wai [arrives at 12:10 EST].

Non-voting Members:

Bob Duff, Niklas Holsti, Brad Moore.

Observers:

Ethan Luis McDonough [leaves 11:00 EST]

Meeting Summary

The meeting convened on Thursday, 11 December 2025 at 10:34 hours EST and adjourned at 13:26 hours EST. The meeting was held using Zoom. The meeting covered all of the regular AIs and one amendment AI on the agenda.

AI Summary

The following AIs were approved with editorial changes:

AI22-0141-1/03 Aspects unspecifiable for classes of types (9-0-0)
AI22-0143-1/02 Make new Decode functions consistent (9-0-0)
AI22-0144-1/03 Classify Data_Error condition in Text_IO.Get (9-0-0)
AI22-0145-1/01 ‘Old should work on values as well as objects (8-1-0)
AI22-0146-1/01 Expected profile of subprogram-valued aspects (8-0-2)
AI22-0149-1/02 Resolution and ambiguity for prefixed views (10-0-0)

The following AIs were discussed and assigned to an editor:

AI22-0131-2/02 Part_Of aspect with aliased results

The following AIs were discussed and voted No Action:

AI22-0131-1/05 Read_Only aspect with aliased results (10-0-0)

Detailed Minutes

Welcome

Steve welcomes everyone to this meeting.

Apologies

John Barnes and Jean-Pierre Rosen said that they will not attend. Richard Wai said that he will be late.

Previous Meeting Minutes

There were no comments on the minutes: Approve minutes: 8-0-0.

Date and Venue of the Next Meeting

Our next electronic meeting is proposed for Thursday, February 19th. There are no objections. So our next meeting is Thursday, February 19th., with the usual time (10:30-13:30 EST [-4 UTC]) and method (Zoom).

Amendment

Randy notes that he has posted a version for comments, and then reposted it after correcting several minor issues noted by Tucker. No one has problems or comments. We vote to send the document to WG 9:

Approve Amendment 1 for Ada 2022 as submitted: 8-0-0.

Prototype parallel implementation

Ethan gives a demo of the parallel features. He shows tests with parallel loops, parallel blocks, and nested parallel loops with exits via gotos. In each case, he shows how the code is expanded into a procedure for the loop body with various calls to the light-weight threading (LWT) library.

Tucker explains that there are two LWT implementations (one based on work-stealing, one based on OpenMP).

Finally, Ethan shows a "real" ray tracing application.

Tucker shows Richard's ACATS test.

Ethan thanks Tucker and Richard for their help developing this. We thank the team (Ethan, Tucker, Richard) for developing this implementation.

Unfinished Action Items

There is no discussion of these items.

Current Action Items

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

Randy Brukardt:

Editorial changes only:

- AI22-0141-1
- AI22-0143-1
- AI22-0144-1
- AI22-0145-1
- AI22-0149-1

Gary Dismukes:

- AI on partial profiles for subprogram-valued aspects (see discussion of AI22-0146-1).

Tucker Taft:

- AI22-0063-1 (Split work with Richard Wai)
- AI22-0071-2
- Revise AI22-0119-1 (see Unfinished Action Items of meeting #63I [May 2025])
- AI22-0131-2
- AI22-0135-1 (AI22-0135-2 is on hold)
- Github Issue #139 (just write response and close)

Richard Wai:

- AI22-0063-1 (Split work with Tucker Taft)

Detailed Review

The minutes cover detailed review of Ada 2022 AIs (AI22s). The AI22s 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 had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the working Ada 202y AARM, the number refers to the text in draft 5 of the Ada 202y AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2022 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2022 AIs

AI22-0131-1/05 Read_Only aspect with aliased results

Tucker wants to abandon this version, so we agree to remove it from future agendas.

No Action: 10-0-0.

AI22-0131-2/02 Part_Of aspect with aliased results

Tucker explains the AI. The idea is to add (compile-time) protection for returning parts of objects, such that the returned objects don't become erroneous because of a change to the enclosing (source) object.

We look at the wording for the "scope of the function call". Randy wonders if this is essentially the same as the point at which the object is finalized. Its always suspicious to have two similar definitions with different wording – one would hope that they work exactly the same. It would be better if the "scope of the function call" piggy-backed on the finalization definition (or vice-versa). [Editor's note: I suspect that making "scope of the function call" the ultimate definition would be difficult, since finalization is based on masters, which are ultimately based on accessibility. OTOH, defining "scope of the function call" in terms of the appropriate finalization of the function result would seem to be easy.]

Tucker notes that this is not intended to handle problems with task interactions. Task abuses typically are erroneous anyway.

Bob asks if these could be used in the existing containers. That's unlikely, as these rules are very conservative and thus it would be too incompatible. For instance, passing a reference as a parameter is going to be illegal with these rules unless the called subprogram has a Global setting (unlikely) or the container is more local than the subprogram (possible, but often false or unknown [a parameter is unknown as to the actual scope]).

Tucker is asked how this differs from the original alternative. This version does not attempt to do anything with tasking, whereas the original alternative did try to handle tasking issues.

Keep alive: 10-0-0.

AI22-0141-1/03 Aspects unspecifiable for classes of types

Steve notes that this AI just defines this new term and does not make any semantic changes. He notes that is is possible that we will have additional uses for this term in the future.

Is "class" the right word? Types in the same class have to be related by derivation. These are formally "categories" (3.2(2)). Tuck changes the wording to use "category", including the subject and title.

Niklas (?) notes that this is placed in the middle of the rules about nonoverridable aspects. Should it be somewhere else? Put this after 13.1.1(18.8/6) (at the end of the Legality Rules section). We discussed putting it earlier, but that would renumber all of the paragraphs about nonoverridable.

Approve AI with changes: 9-0-0.

AI22-0143-1/02 Make new Decode functions consistent

Randy notes that the AI introducing the Decode functions was the only one proposed for the Amendment that he hadn't yet added to the RM. So of course there was a problem: the new functions were handled differently than the existing ones. The existing ones had individual descriptions of how they work, while the new ones just had a general overall description.

This AI makes the definitions match the original functions. Randy notes that he moved a paragraph into the individual descriptions; it is duplicative, but it also eliminates almost half of the words which were about identifying the functions and parameters involved. (Which are not necessary with the specification directly above the wording.)

This AI was included in the Amendment in order that the printed Ada-Europe version looks consistent.

Brad noted a typo that was fixed in the Google Docs version.

Approve AI with changes: 9-0-0.

AI22-0144-1/03 Classify Data_Error condition in Text_IO.Get

Randy notes that this topic has been raised many times in the past, but it was left unfixed as getting the right wording was hard.

Niklas suggested a rewording of the text which seems a bit clearer. We agree to use it, Tucker applies it to the Google Docs version.

We discuss a comment on the AARM note. While Tucker and Randy think it is fine, others think it isn't. The text is changed slightly in the Google Docs version.

Christoph is concerned that the word "prefix" has a technical meaning, and would prefer something like "initial fragment". Bob thinks "prefix" is better fit in English. He notes that we can use the same word for multiple purposes (task dispatching vs. call dispatching). Discussion leads to no clear conclusion, so we take a straw vote.

Straw vote: "prefix" (9); "initial fragment" Jeff, Jean-Pierre, Christoph (3), abstain: 0.

We leave the use of "prefix" as is.

Approve AI with changes: 9-0-0.

AI22-0145-1/01 'Old should work on values as well as objects

Tucker explains the (simple) change. There doesn't seem to be any semantic reason to restrict this to objects. He notes that there is a simple use for My_Arr'Length'Old. One could write My_Arr'Old'Length instead, but if My_Arr is large, that would be much more expensive at runtime.

Randy asks that the discussion of removing distinctions between object and value be removed. He thinks it is unrelated musings for this AI. He also states that he thinks it would be a massive amount of work. (Tucker had suggested that the term "object" be removed in favor of "variable", such that there would only be variables and values; while that might be a better model, there is a lot of text that talks about objects in Ada currently. Given that we've already removed the distinction for renamings and from the prefix of 'Old, there is not a lot to gain from removing it in other cases.)

Randy wonders if the constant is well-defined for a named number. (Of course, such a usage is not useful, as the value can't change during the subprogram call.) The type is a universal type. Indeed, the possibility of the type being universal is mentioned in passing in 6.1.1(26.9/4). Someone asks how this is represented at runtime ('Length does not have to be static); 4.4(10.1/5) says universals are represented as *root_integer* or *root_real* as appropriate at runtime. That certainly applies here.

Approve AI with changes: 8-1-0. Justin opposes (thinks it is overly complicated and too hard to implement).

AI22-0146-1/01 Expected profile of subprogram-valued aspects

Gary tries to explain the source of the AI. Randy then takes over and explains that we have the rules necessary to define what is Name Resolution and what is not. So he wrote this AI to confirm the intent in general, without making any changes to the intended semantics.

In the specific case of indexing, the original AI is quite clear that the intent is to include all functions with the correct name, thinking that otherwise it would be difficult to determine what indexings are defined. The fact that the profile is described in two separate sentences also adds to that (the first was intended to be resolution, the second was intended to be a legality check). The sentences could have been easily combined, they were written separately for a reason. So he added some wording to clarify what is used for resolution.

Tucker asks about the AI22-0084-1 added rule, he thinks it is too narrow. He says that generic formal parameters can be picked up as the rule is written. Bob asks why the rule doesn't say "immediately within". Randy thinks that we wanted to allow routines declared in nested packages to be usable as aspects. In any case, we agree that this question seems out of bounds for this AI.

We have some discussion on whether we need to define a new kind of resolution to deal with cases where only a partial profile is required. It would be best to see some examples as to the effects of that. We ultimately agree that this sort of change needs a different AI.

We assign a new AI on partial profiles, and possibly generic formal parameters to Gary.

Approve AI with changes: 8-0-2. Steve, Gary abstain.

AI22-0149-1/02 Resolution and ambiguity for prefixed views

Randy tries to explain the ambiguity problem and the split the baby fix. The proposed rules try to preserve the automatic dereference for selected notation (and the compatibility with existing rules), while also eliminating the anomaly of a name that can be used on a partial view not being available for the full view.

In particular, these rules eliminate the possibility of a legal prefixed view having one meaning in the client view and a different one in the package body (and child units). Randy notes that this could be a significant problem in unit tests that are built as child units – unit tests typically want the client view for most purposes. Also, the name in the body does not reference the subprogram declaration given clearly in the specification, but rather one from a different package far away. That's likely to be very confusing. The rules as proposed make such a call ambiguous and thus illegal.

Bob notes that this situation is similar to the Beaujolias effect, which we take great care to avoid. Randy notes that use-clause conflicts are a similar issue with a similar solution.

Randy notes that this is the worst of both worlds from an implementation perspective, since it will have to try both with and without the implicit dereference (if only one works, that is an appropriate solution).

It is noted that the default expression would not conform with the current rule, since there is a name embedded within the expression that designates two different declarations.

Bob would like an AARM note to say that one may need to try it both ways. That seems reasonable.

Tucker would like to change the “after any implicit dereference”. He thinks it can be read in both ways (he has always read it as a requirement, but he understands how it looks very different than the obvious requirement for attribute prefixes).

A suggestion is (“with or without an implicit dereference”).

Gary would like to summary to be clearer. Tucker rewords.

Tucker objects to “wrong” in the !issue (because he interprets the wording differently), and we change that.

Approve AI with changes: 10-0-0.