

## Minutes of ARG Meeting 63J

30 July 2025

### Attendees:

### Voting Members:

Steve Baird, John Barnes, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Ed Fish, Christoph Grein, Niklas Holsti, Brad Moore, Alejandro Mosteo, Jean-Pierre Rosen (joins 11:19 EDT), Justin Squirek, Tucker Taft, Tullio Vardanega, Richard Wai.

### Non-voting Members:

None.

### Observers:

Robin Leroy (Unicode liaison), Ethan Luis McDonough (US) (leaves 11:45 EDT)

## Meeting Summary

The meeting convened on Wednesday, 30 July 2025 at 10:38 hours EDT and adjourned at 13:33 hours EDT. The meeting was held using Zoom. The meeting covered all of the new and updated AIs on the agenda.

### AI Summary

The following AIs were approved:

AI22-0137-1/02 Renaming-as-body with a parameter of an incomplete type (15-0-1)

The following AIs were approved with editorial changes:

AI22-0075-1/06 Explicitly Aliased Results (16-0-0)

AI22-0138-1/01 Inheritance and conformance (14-0-2)

AI22-0139-1/01 Aggregates and build-in-place (16-0-0)

The intentions of the following AIs were approved but they require a rewrite:

AI22-0140-1/03 Requiring unconstrained subtypes (16-0-0)

The following AIs were discussed and assigned to an editor:

AI22-0076-2/01 No\_Dynamic\_Accessibiliy\_Checks restriction

AI22-0129-1/03 Usage example for Protected Type with Dynamic Predicate

AI22-0131-1/01 Read\_Only aspect with aliased results

AI22-0135-1/02 Coextensions, functions calls, and storage pools

AI22-0141-1/01 Aspects unspecifiable for classes of types

## Detailed Minutes

### Welcome

Steve welcomes everyone to this meeting.

### Apologies

Jean-Pierre Rosen notes that he will be late. Randy notes that when Jean-Pierre arrives, all active voting members of the ARG will be present at this meeting.

### ***Previous Meeting Minutes***

There were no comments on the minutes: Approve minutes: 15-0-0.

### ***Date and Venue of the Next Meeting***

Our next electronic meeting is proposed for Wednesday, October 8. Several people note that is the day of the WG 9 meeting (oops, says Randy). October 9<sup>th</sup> is suggested. Several people think they have conflicts and then decide it is fine. We agree on October 9<sup>th</sup>.

Later, John announces that the 9<sup>th</sup> does not work him. We discuss the 16<sup>th</sup> and 7<sup>th</sup>. The 7<sup>th</sup> is the Ada User Society meeting. (Both Randy and Richard wonder why, as sponsors, they don't know this.)

So we change to October 16<sup>th</sup>. Richard can't make that. More discussion leads to choosing Monday, October 6<sup>th</sup>, with the usual time (10:30-13:30 EDT [-5 UTC]) and method (Zoom).

### ***Format Updates on older AIs***

Randy reports that he has finished permanently reformatting all of the AIs that were original in text form to HTML.

He has turned to looking at AIs that have unnecessary differences between versions (Bob complained about that during the last meeting). Various converter updates in 2023 and recently have reduced these to very few, however taking advantage of that requires creating new archives containing newly converted versions. Randy says that he has been keeping the original Google Docs outputs of every version so that we can do this sort of reconversion if necessary. He is planning to concentrate on the AIs for which such work would have the most positive effects, since it would be impractical to redo all 101 remaining AIs.

### ***Progress report on the Corrigendum***

WG 9 has voted to approve and advance the Corrigendum. So our work on that should be complete, pending results at higher levels.

### ***Themes for upcoming work***

Tucker has created a draft "themes" for future work. He goes through them.

Steve thanks Tucker for doing this. Various others second that during subsequent discussion.

Bob notes that Tucker mentioned "lock-free" during his verbal description, but didn't mention that in the text. It is suggested to make some of these less prescriptive, so the ARG can decide whether or not to do some of these things.

In particular, several people express reservations about the idea of adding memory models to Ada; that seems too tied to a particular hardware implementations and can be difficult to get correct.

Bob suggests that string handling mentions IO (in particular, a new version of Text\_IO).

Ed mentions that some work was already done on the generic proposals in various proposed AI12s.

Steve asks whether we want to see this again. We can't wait until our next meeting, as the WG 9 meeting is before that (we later changed the date of the meeting, but it still would be too close as WG 9 members would have no time to review it). Tucker will update and send around (on the mailing list) a new version by September 1st.

### ***Parallel implementation***

Ethan gives a report on the implementation of the parallel features (part of "summer of code"). He's completed parsing and semantics for parallel loops and parallel do, and the expansion of the sequential fall-back version. He's still working on the actual parallel expansion.

He shows some examples.

We are happy to see this progress in a critical area.

### ***Unfinished Action Items***

Randy asks Tucker why his homework wasn't done on time. He says that he had the wrong week for the deadline. Randy notes that he compounded that by forgetting to send a reminder before leaving on vacation. Steve also did some work after the deadline.

### ***Current Action Items***

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

Steve Baird:

- AI22-0076-2
- AI22-0141-1

Randy Brukardt:

- AI22-0140-1

Editorial changes only:

- AI22-0075-1
- AI22-0138-1
- AI22-0139-1

Christoph Grein:

- AI22-0129-1 (with assistance from Tucker Taft)

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])
- AI22-0129-1 (assist Christoph Grein)
- AI22-0131-1
- AI22-0135-1

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 3 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***

#### **AI22-0075-1/06 Explicitly Aliased Results**

Tucker shows the example of how this works. We discuss the accessibility check briefly.

Gary would like some uses of functions returning explicitly aliased results in the !recommendation.

It is suggested to use a target name symbol in the last line of the !example:

```
C (L) := @ + 1;
```

Do we believe this needs prototyping? Tucker says that this is just compile-time checks (no changes to the actual runtime code), and he doesn't think those ever need prototyping. And he's implemented something like this in Parasail. No one objects to his analysis.

Steve asks if there are any interactions with generics. Such as returning a component of generic **in out** formal parameter. We try to figure out what 'Access rules are for such a case. Ed thinks that the actual object has to outlive the instance (as it is passed in), so there wouldn't be an issue.

Randy thinks that 'Access probably has handled most such issues, so we are unlikely to have a new problem from this construct.

Approve AI with (future) changes: 16-0-0.

Randy later asks if the 'Constrained issues have been addressed. Yes, but it is dynamic (the value of Result'Constrained is returned along with the object); we need the AI22-0140-1 mechanism to eliminate the overhead for mutable types.

### **AI22-0076-2/01 No\_Dynamic\_Accessibiliy\_Checks restriction**

Randy notes that the previous attempt was in alternative 2, and it contains wording. Thus, Steve updated the wrong alternative (and didn't update the proposed wording), so we don't have a consistent set of rules to review. We decide to defer this one until that is fixed.

### **AI22-0129-1/03 Usage example for Protected Type with Dynamic Predicate**

Tucker was supposed to update this, but didn't, so we skip it.

### **AI22-0131-1/01 Read\_Only aspect with aliased results**

This aspect prevents changes to a variable within the (dynamic) scope of a related object.

```
C : Obj renames A.B with Read_Only => A.X;
```

Bob asks if a full assignment to A would be allowed if it doesn't change the discriminant value. Tucker's model is that this does not cause any runtime checks, so that can't be allowed.

Steve asks what happens if one takes 'Access before declaring C, how can one prevent an assignment to A via that other path. Tucker says that aliasing needs to be taken into account: this check is *very* conservative. For instance, modifications via "**access** Disc" would have to be disallowed (since we can't know if that points to A).

Ed asks about tasking. Tucker notes that direct references from multiple tasks would be erroneous for other reasons unless synchronized. Steve wonders if this prevents "proper synchronization" of these objects, because that would introduce problems with this compile-time model.

Jean-Pierre notes in a protected object, this would work fine. But Randy notes that using atomics to synchronize would eliminate the erroneousness, but potentially cause problems. As Steve notes, the current rules make problems impossible, but here we possibly are opening a door.

More work is needed on this one, to define a set of rules.

Gary wonders if these renaming cases are interesting enough to spend effort on. The ultimate goal here is to use this for container references (especially via aliased function returns); the renaming case is easier to understand because

the complications are not hidden like they are in the containers. So it makes an instructive example even if it is not very interesting by itself.

Tucker notes that many uses of indexings are very short-lived, so the effect of these restrictions would not matter much. However, if an indexing is passed as a parameter or renamed, it could last a long time in which case these sorts of rules are needed.

The Constrained aspect (as defined in AI22-0140-1) could be used to ensure that the discriminant can't be changed via an access value. Such assignments could be allowed since they can't change the discriminant in question.

Steve notes that there are other rules that say that something has the restrictions of a renaming. Randy says that an `iterator_specification` is a specific example of such a thing (there are a number of others). We need to ensure that those cases still work if we weaken the restrictions in this case.

Keep alive: 15-0-1. Abstain: Jeff.

### **AI22-0135-1/02 Coextensions, functions calls, and storage pools**

Tucker would like to rewrite this; it failed its vote last time. It doesn't make sense to revote it without changes. (Continually bringing it to a vote until the right mix of members is present to allow it to pass is not the way to build consensus!)

Ed moves to defer this one (because the discussion was devolving into minutia of how it might be rewritten, and this is not the forum for that sort of discussion). There is general agreement with this motion and we move on.

### **AI22-0137-1/02 Renaming-as-body with a parameter of an incomplete type**

We want to allow implementations to use a wrapper model for such bodies. We don't want to require writing wrappers that would be illegal if written explicitly.

Gary wonders if this is incompatible. Not for GNAT, it generates the wrapper and then rejects it as illegal. So no one could have GNAT code that uses it. Tucker notes that it seems impossible to avoid building a wrapper in all circumstances; when bodies are separately compiled, there has to be a body generated somewhere. So it seems unlikely that any compiler fully supports these cases.

Approve AI: 15-0-1. Christoph abstains.

### **AI22-0138-1/01 Inheritance and conformance**

Randy explains the problem. Private extensions and their full types inherit subprograms from their ancestor separately, but conformance requires the entities to be declared by the same declaration. That can't happen for function calls in the spec and body for a private extension, as they necessarily come from different inherited declarations.

Randy did a compiler survey using what he hopes will be a future ACATS test. GNAT does something weird (allowing declarations to conform even when they have different parameter names or default expressions). Janus/Ada and ObjectAda implement the rule literally (rejecting all of the default expressions as being non-conforming).

We considered simply confirming the language, but it seems very unfriendly for users to have inherited subprograms for a private extension and the corresponding full record extension to not conform when they inherit from the same exact declaration. Charitably, we can presume that GNAT was trying to avoid this situation and got it very confused.

Thus, he proposes to extend the conformance rule to allow inherited routines to conform so long as the ultimate non-inherited ancestor routine is the same.

John would like the example to replace "Cnt" with "Count".

Approve AI with changes: 14-0-2. Christoph, Jean-Pierre abstain.

### **AI22-0139-1/01 Aggregates and build-in-place**

Steve notes that building a large aggregate in a tiny space when “build-in-place” is required is not going to work. Randy notes that it is possible to implement using heroic efforts, in particular, the evaluation of each expression has to check whether it fits before writing it, and if it does not, evaluate it into a temporary object which is then discarded. But requiring that seems silly when simply checking sooner avoids the problem.

Randy says that he used an implementation permission because he doesn’t have to be as precise about what items have to be evaluated before the check (the discriminants, and some of the array indexes) and which ones can be evaluated afterwards. Also, this matches what was done for function returns, it is not specific to build-in-place (which is implementation-defined in many cases anyway), and if the implementation did already go to the heroic efforts described previously, it does not require it making any changes.

Steve wonders if we need to say that implementations cannot trash memory. Tucker says that we don’t have to tell implementors not to do stupid things. Randy says that is covered as implementations cannot introduce erroneous execution where it is not explicitly permitted, and nothing is permitting it here. Tucker says he agrees with Randy’s take.

Gary asks that a hyphen be removed from “non-discriminant”.

Approve AI with changes: 16-0-0.

### **AI22-0140-1/03 Requiring unconstrained subtypes**

Randy explains the basic idea. We have a new subtype property, Constrained, which has the effect of determining whether an object can have its discriminants changed. We use Legality Rules to ensure that any objects converted to a subtype that specifies that objects can have their discriminants modified in fact are unconstrained.

This can remove runtime overhead from parameter passing (since the Constrained value becomes known at compile-time), eliminates a tripping hazard (passing a constrained object to a routine that intends to change the discriminants becomes illegal), and fixes the problem with Variable\_Indexing necessarily returning a constrained-by-its-initial-value that cannot have the discriminants modified.

Ed asks how that ‘Constrained attribute interacts with this. The ‘Constrained attribute returns True or False based on this nominal subtype; only if the Constrained property is Unspecified does it revert to the possibly runtime decision required by Ada since the beginning (Ada 83). Randy notes he needs to add wording to that effect.

Randy tries to explain the model of False\_if\_Actual. The idea is that the Constrained aspect value gets set appropriately in an instance based on the actual subtype for the formal type used in the subtype. We do a bit of wordsmithing.

Steve notes that Gary had a different solution, putting a flag on access declarations. He says that would be much simpler. Randy notes that an objection is that to do that on anonymous access types, one would have to put the aspect on the “wrong” entity, since the anonymous type itself cannot have aspects. It also doesn’t address the parameter case.

Tucker thinks that the parameter case is important, and thinks it should be addressed.

We later note during the discussion of AI22-0075-1 that it too has a dynamic overhead for ‘Constrained associated with the aliased result. This mechanism also could be used to avoid that overhead (and possibly reintroducing the problem with Variable\_Indexing returning constrained objects).

Approve intent: 16-0-0.

## **AI22-0141-1/01 Aspects unspecifiable for classes of types**

This AI was submitted after the deadline, so we won't make a final decision on it at this meeting. A review and discussion still can be valuable, and we only have a little time remaining today.

Steve explains the problem. Some aspects work on all types except one that “naturally” has the associated property (integer types have integer literals, array types have indexing, and so on). We had originally handled that by saying that the aspect is not specifiable on such a type. However, we need to recheck that rule in generic instances, lest a descendant of a generic private type define such a property, and then the actual type be one that already has the property. For that to happen, it has to be described as a Legality Rule (definitions are not rechecked).

Steve folded all of the rules into the term “unspecifiable” so that they don't need to be repeated (and sometimes forgotten). That includes the rule that the full type of a private type cannot be in the unspecifiable class.

Tucker dislikes defining a term for this. Randy notes that was his initial reaction as well, but the elimination of the existing rules in favor of this bundle changed his mind. He notes that fixing the next such problem discovered will be a lot easier if there is a term. (He has little doubt that we eventually will find another problem here.)

Ed wonders if a meta language would help here, with a more hierarchical organization of types. Randy notes that these user-defined things tend to cross-cut a hierarchy, applying to anything but certain items in the hierarchy. That “but” leads to these sorts of rules, and in particular the need to recheck the rules in generic instantiations.

Keep alive: 14-0-2. Abstain: John, Justin.

[Editor's note: No specific instructions were given to the author for this one, so any update would depend on review comments. It would be fine to bring this back unmodified in the absence of such comments.]