Minutes of ARG Meeting 63N

12-13 May 2026

Attendees:

Tuesday Session:

Voting Members:

Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Christoph Grein, Niklas Holsti, Brad Moore, Jean-Pierre Rosen, Justin Squirek [arrives 12:50 EDT], Tucker Taft, Tullio Vardanega.

Non-voting Members:

Bob Duff, Alejandro Mosteo [left 12:16 EDT]

Observers:

Robin Leroy (Unicode liaison) [arrives at 11:30 EDT]

Wednesday Session:

Voting Members:

Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Niklas Holsti , Brad Moore, Jean-Pierre Rosen, Justin Squirek, Tucker Taft, Tullio Vardanega, Richard Wai [left 13:00 EDT].

Non-voting Members:

Bob Duff, Alejandro Mosteo [left 12:58 EDT]

Observers:

Robin Leroy (Unicode liaison) [arrives at 11:17 EDT]

Meeting Summary

The first session of the meeting convened on Tuesday, 12 May 2026 at 10:32 hours EDT and adjourned at 13:32 hours EDT. The second session of the meeting convened on Wednesday, 13 May 2026 at 10:34 hours EDT and adjourned at 13:27 hours EDT. The meeting was held using Zoom. The meeting covered all of the AIs on the agenda as well as several Github issues.

AI Summary

The following AIs were approved:

AI22-0151-1/04 Syntax rules disagree with Text_IO.Get results (8-0-2)
AI22-0154-1/05 Revised resolution of indexing aspects (9-0-1)

The following AIs were approved with editorial changes:

AI22-0157-1/08 Fix three holes in definition of “known on entry” (10-0-0)
AI22-0159-1/01 Inheritance for aspect allowed to denote multiples subprograms (9-0-1)
AI22-0160-1/01 Improved variable-vs-constant indexing (10-0-0)

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

AI22-0075-1/12 Explicitly Aliased Results (10-0-0)
AI22-0076-2/04 No_Dynamic_Accessibility_Checks restrictions (9-0-1)
AI22-0131-2/04 Part_Of aspect with aliased results (10-0-0)

AI22-0147-2/01 Controlling what is a controlling parameter (8-0-2)
AI22-0148-1/03 Bounded and unbounded indefinite objects (11-0-0)
AI22-0152-1/07 Optional objects (10-0-1)
AI22-0161-2/02 Universal Unicode Text (11-0-0)

The following AIs were discussed and voted No Action:

AI22-0147-1/02 Controlling what is a controlling parameter (10-0-0)
AI22-0161-1/02 Provide Unicode and universal Text interfaces (11-0-0)

The following Github Issues were discussed and assigned to an editor to create an AI:

#95 'Value for enumerations
#137 Max_Stream_Size for composite types?
#156 Suggested enhancements to the wording in 3.10.2 about "tied" accessibility levels
#170 13.11.5 Subpool Reclamation: Is it the compiler's or user's responsibility to prevent loops?
#173 Illegal class-wide preconditions on Allocate/Deallocate_Subpool in 13.11.4

The following Github Issues were discussed and assigned to an editor to close:

#111 size of immutably limited unconstrained discriminated components
#119 Unchecked_Deallocate_Subpool should be preelaborated.

Detailed Minutes

Welcome

Steve welcomes everyone to this meeting.

Attendance

On Tuesday, Randy takes attendance. 10 voters, 2 non-voters, and 0 observers. (Another voter and an observer showed up later; a non-voter left early.) On Wednesday, 11 voters, 2 non-voters, and 0 observers. (Another voter and an observer showed up later, a voter and a non-voter left early.)

Apologies

John Barnes will miss both days; Richard Wai said he will be late Tuesday (he didn't attend at all on Tuesday); Bob Duff said he will be late Wednesday (but he arrived just as we were starting); Christoph Grein said he has to travel on Wednesday and cannot attend; Justin Squirek said he will be late on Tuesday. Observer Robin Leroy said he would be late both days (although that message wasn't received until after he arrived on Tuesday).

Previous Meeting Minutes

There were no comments on the minutes: Approve minutes: 10-0-0.

Date and Venue of the Next Meeting

Our next electronic meeting is proposed for July 21-22nd. We wait quite a while for someone to object to those dates, and no one does. So we adopt those dates. Afterwards, Jeff noted that he has a bowling match ("not tenpin") on the 21st so he will be late (but he didn't ask to change the date).

Unfinished Action Items

We don't discuss any of these items. Steve notes that he has two unfinished items assigned after the last meeting (a minor wording fix for the default image, and a fix for conflicting aspects when otherwise identical routines are inherited).

Current Action Items

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

Steve Baird

- AI22-0076-2 (new alternative building on the proposed Static_Checks pragma).
- Wording fix for default images involved qualified expression syntax (from private mail)
- Fix for conflicting aspect when otherwise identical routines are inherited (from private mail)

Randy Brukardt:

- AI22-0161-2
- AI for Static_Checks pragma (see Github #167)
- Github Issue #111
- Github Issue #119
- Github Issue #156

Editorial changes only:

- AI22-0157-1
- AI22-0159-1
- AI22-0160-1

Niklas Holsti:

- AI22-0147-2

Brad Moore:

- Github Issue #170
- Github Issue #173

Justin Squirek:

- Github Issue #95

Tucker Taft:

- AI22-0063-1 (Split work with Richard Wai)
- AI22-0071-2
- AI22-0075-1
- 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)
- AI22-0148-1
- AI22-0152-1
- AI22-0153-1
- AI22-0158-1
- Github Issue #137
- Github Issue #139

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

AI22-0075-1/12 Explicitly Aliased Results

Tuck explains that the dynamic semantics were updated. Niklas had noted that the existing dynamic semantics did not make sense for aliased returns. Tuck admits that this was not his best work (missing an entire set of rules that needed changes).

Tuck notes that we haven't resolved the issue of subtype matching, Randy had noted that is very restrictive: no discriminant-dependent array component could be returned that way. For a read-only reference, it makes sense to be less restrictive. (For a read-write reference, it is best to match exactly as otherwise the checks needed get very complex.)

Steve worries about returning a discriminant-dependent component of an unconstrained object passed to a constrained parameter. (Since the parameter is constrained, the component can be renamed and thus is legal to return.) If the actual parameter is updated after the call but before the return object ceases to exist, the returned component might disappear. Randy points out that this is covered by 3.7.2(4) which makes such cases erroneous, even for normal parameter passing. This case is essentially similar to an access return of the same component, which has been possible in Ada since Ada 2005. Unfortunately, it's not possible in general to detect all such cases, even if one ignored all discriminant constraints (which would have an undesirable effect on performance).

Approve intent: 10-0-0.

AI22-0076-2/04 No_Dynamic_Accessibility_Checks restrictions

Randy is asked to explain the suggestion about static checks. We would like to be able to specify that specific kinds of checks are checked statically (meaning the program is illegal if they fail). For that, we need a list of kinds of checks, and we already have a complete such list in 11.5. That led him to propose that Static_Checks is essentially a safer form of Suppress; rather than being erroneous if a check if suppressed incorrectly, the program is illegal. That also gives us a local way to allow a dynamic check (probably in the scope of a local handler, as in a saturation math package). As a form of Suppress, it would work on all kinds of checks, with a minimum requirement for canonical checks that can be eliminated (some kinds, like Storage_Error, cannot be practically eliminated, so Static_Checks (Storage_Check) would almost certainly render a unit illegal). We haven't written an AI for this yet.

Tucker explains that he believes that this restriction should really define additional static checks, and whether the dynamic checks are eliminated is left to the compiler. Written that way, the restriction does not need to be partition-wide (although that needs to be a possibility for ideal optimization). [Editor's note: I had privately noted that this does not sound like a Restriction to me; we explored Static_Checks in part for that reason.]

Steve wants to vote intent that we make an alternative proposal on the lines that Tucker suggests. We will still need most of these rules (either as a separate restriction, or associated with something like Static_Checks), but not some of the detailed discussion of dynamic checks.

Approve intent: 9-0-1. Christoph abstains.

AI22-0131-2/04 Part_Of aspect with aliased results

Tucker notes that he made some minor changes to the AI. The idea is that if you indicate that the result is part of a parameter, then you can't do various things with that parameter so long as the function result is in use.

The key is that the rule about renaming a part of an object is relaxed when you have a Part_Of on the object.

Steve noted that this could have the effect to ensure that the result has the accessibility of the parameter. That would make user-defined indexing work more like arrays. Tucker will look into that.

Tucker wonders if this should work on reference returns as well as aliased returns (it seems like it should). He will add that.

Approve intent: 10-0-0.

AI22-0147-1/02 Controlling what is a controlling parameter

Tucker notes that this type-based mechanism doesn't work well for conformance. T and T'Rooted match the same types – they really should conform somehow.

He has concluded that this is the wrong solution.

We agree that if the author wants to abandon the AI, an immediate No Action vote is appropriate.

No Action: 10-0-0.

AI22-0147-2/01 Controlling what is a controlling parameter

The idea is to use a marker (currently the keyword **abs**) to mark parameters or results that should not participate in inheritance. The operations would remain primitive, so that prefix notation and similar rules would continue to work.

For overriding, the **abs** would need to match. It certainly needs to be in "full conformance". It probably should be ignored for subtype conformance and mode conformance. We need a special rule to not ignore it for overriding.

Gary wonders if we could use an aspect specification to specify this. That seems to make the specification longer, not necessarily clearer.

abs comes before the mode and so on. Same with **return**.

Steve suggests that it acts like an anonymous one component record wrapping the parameter.

This is intended to work for all types. The subject implies it only is for tagged types, as there are no controlling parameters for untagged types.

The predefined operators would be function "=" (Left, Right : Integer) return abs Boolean; These don't inherit for Boolean as they are defined specifically for each type (and are not inherited). We probably wouldn't need that mechanism anymore, but changing it would be more complicated than any gain.

abs means "absolute" or one of a zillion other words that Tucker mentions that the editor has never heard of. (I think he was reading from a dictionary.)

We'll look for a better marker.

Approve intent: 8-0-2 Gary and Randy abstain.

AI22-0148-1/03 Bounded and unbounded indefinite objects

Randy notes that he is working on an implementation report for this AI. He notes that individual objects don't need to be controlled in order to implement this (cleanup can be scope-based; technically, associated with masters).

Tucker notes that with this change, many of the existing restrictions associated with indefinite objects are no longer true.

Do we need an attribute to check the extent? If so, where does the enumeration go? Tucker did not create such an attribute at this time, in part because he didn't want the enumeration in Standard and there is no other obvious place.

[Editor's note: System might be an appropriate place for such an enumeration; memory management certainly is a system function. An attribute might be needed for a role similar to 'Constrained, particularly in generic units where the details are not known statically.]

Steve notes that Ada makes a distinction between finalize and cease-to-exist. This is unlike a holder which finalizes and frees the memory at the same time. Tucker notes that a holder is by definition an optional object (it starts out containing no object at all). Randy notes that there is no obvious new implementation problem; this is already true for any object. His model is that there is a storage pool associated with each master, and it is defined first (so it is cleaned up last). In this way, all other finalization is done before any heap-allocated memory is freed. (Any memory that was able to be allocated from the stack frame of course also does not get freed until all other operations are complete.)

Jean-Pierre notes that we need a more convincing example, one where we don't have an existing solution. Gary would like a more elaborate example. Brad suggests that an example of bounded objects is needed.

Gary wonders if these need a restriction on renaming, as some of the components can disappear on assignment. It seems necessary to prevent renaming of subcomponents of one of these unconstrained objects.

Bob says that "it seems like a nuisance to not be able to specify an extent or bound on a subtype".

What does it mean if you use it on a subtype? Does it change streaming? (Subtypes don't affect streaming.)

Steve asks about the interaction between bounded objects that contain unbounded subcomponents. That can lead to fragmentation and probably should be avoided in some uses. (If the object needs to grow, there may not be enough memory to do that. Indeed, even if there is sufficient memory to hold all of the component's current size, the way they got there might mean that there is not enough memory for the components.) That should be explained in an AARM note or perhaps a Usage Note.

Tucker asks if "Bound" is the right term. Containers use the name "Capacity" for this. Niklas notes that Capacity is in terms of logical elements, not storage elements. We reach no conclusion on this.

Approve intent: 11-0-0.

AI22-0151-1/04 Syntax rules disagree with Text_IO.Get results

Christoph explains why he believes case 2 should not ignore trailing blanks.

Additional justification for this rule was added to the !discussion since the last meeting. Randy had noted how the existing wording talks extensively about skipping leading blanks and is conspicuously silent about trailing blanks. Christoph mentions how almost all of the compiler implementers reject trailing blanks, and also how doing so makes reading columns of numbers safer.

Approve AI: 8-0-2. Steve and Gary abstain.

AI22-0152-1/07 Optional objects

Tucker has added wording, and additional rationale to the previous version.

He suggests replacing various possibly confusing terms "null string", "null array", "null range". Some worry about the churn involved in that, as the terms (particularly null string and null range) are embedded in tools and books and other infrastructure.

He also suggests replacing as "null constraint" (which is already confusing as it sounds like it has something to do with null exclusions). Randy would prefer a different term than "empty constraint", as this is no constraint at all. Whatever we pick should not sound like a form of constraint.

He introduces the idea of optional types and objects in 3.2 "Types and Subtypes".

He added null_inclusion to the type declarations as requested; he doesn't modify the type definitions themselves.

Randy asks why derived types do not allow null inclusions. Tucker says that the parent subtype indication is a better place for the inclusion indicator.

A lot of meat is in 3.2.2 "Subtypes".

Jean-Pierre objects to having subtypes with an additional value. Tucker claims that a null inclusion creates a new subtype that is a union of two subtypes. He notes that we already have 'Base that gives potentially more values. Steve doesn't think that is true example (the base type always has the values, this just exposes them); he thinks this is still a new concept.

By default, the first subtype does not include null (the type always includes null) except for access types.

Tucker notes that integer types include all of the values of integer, regardless of the range of the first subtype. We get bogged down on those details.

It seems that saying the type always includes null is a more sensible model than talking about a "union of two subtypes". Just say the subtype is exposing a capability of the underlying type; that's what happens for 'Base. So this is not a new capability.

You can specify null_exclusions only if null is included; null_inclusions can be specified only when null excluding and null indeterminate. We want to avoid excluding null multiple times.

Incomplete types are discussed. Randy notes that we don't allow components of incomplete types, Tuck says we want to do that for recursive types (but only when the type is completed in the same scope). Randy wonders why as you can directly give the type needed. One would need that capability for mutually recursive types.

That wording is missing; it would go somewhere in 3.10.1.

Steve objects to the wording 3.3.1(11), it is clarified.

The default is null for optional objects, as the default is that the object is not there. Other default values (like Default_Value for scalars) only apply for non-optional objects.

Tucker wonders if Default_Component_Value should be made illegal if the components are optional (it could never be operative).

Steve wonders if a scalar discriminant can include null. You can't use that as an array index or case statement, but it still could be used as another discriminant.

A case statement could allow null if **null** is allowed as a choice. We would need to define that. It does need to be disallowed as an array index value.

Steve wonders if this is different from "**case** P.**all is**", that doesn't fall into the **others** limb (it raises Constraint_Error). He thinks that this might be error-prone.

He thinks you could have a compile-time rule to explicitly mention **null**. Tucker doesn't think falling into **others** is a problem. Steve says it is important that this is a deliberate decision. Note that **others** currently has to stand alone, requiring or allowing **null** to stand next to it would be a change.

Randy wonders if we need universal access at all anymore, as **null** no longer has that type. Tucker replies that we still need universal access as it mostly has to do with equality for anonymous access types.

Steve wonders about static expressions that include null. He thinks that works as a runtime check failure makes an expression non-static. Randy notes that **null** should be static for scalar types. Tucker says strings probably should allow that as well. Some text is needed in 4.9 to accomplish that.

Randy notes that the "single type" rule for **null** is incompatible on the margins, as universal_access can resolve in type conversions and the like. That is unlikely, but possible.

Tucker explains this resolution rule in 6.4.1(3), that should eliminate most incompatibility. Gary notes that formal subprograms don't necessarily know whether the parameter subtype includes null. Matching only requires mode conformance.

Tucker thinks that it might be necessary to include null inclusion as part of mode conformance. Steve notes that we wish Ada 83 had tighter rules, as this is new we can do that.

Renaming and generic formal subprograms have special rules for null exclusion already; something like that is needed at a minimum for null inclusion. The idea of those rules is that there is no lying; that is, if something explicitly says it excludes null, then the actual better do the same, but if it isn't mentioned, then the formal can go either way. Randy notes that writing tests for that and getting those rules to work was by far the hardest part of implementing null exclusions in Janus/Ada. (And the resulting tests gave a lot of trouble to other implementors as well.)

Steve complains about a lack of granularity for Null_Check. If you don't want to suppress access checks, just optional component checks, you can't do that. Jean-Pierre notes that you could Suppress(Null_Check); Unsuppress(Access_Check); to just suppress the new checks (so you can do that). Randy notes that the proposed Static_Check pragma might want more granularity; requiring no checks fail is a hard thing to accomplish and we wouldn't want to have to do that for unrelated uses.

What are the implications of a null task or null protected object?

Finalization of a null object does nothing, even if it is a task or controlled or protected. That needs to be said explicitly in 7.6.1. Assignment of the null value does not call Adjust. We need wording to say that too.

Subcomponents of optional objects are not legal for renaming. We need wording for that.

Steve wonders if we try to dispatch on null as a controlling parameter. Tucker notes that controlling access parameters have to be null excluding; it seems that controlling parameters need to be the same. We don't have the compatibility issue that required us to assume that controlling access parameters are automatically null excluding, so we probably want to make it illegal to specify a controlling parameter of a null including subtype.

We talk briefly about C compatibility: what if a null including type is specified as a parameter for a C function? Probably that is implementation-defined, it might make sense to do something specific, but a general rule doesn't seem appropriate at this time. Niklas notes that we could assume that such a thing corresponds to a C pointer (like we do with **in out**). That doesn't seem right for scalar types. If a conventional null excluding type is used, an appropriate check would occur on the Ada side before the call, so nothing is needed for that. [Editor's note: But what about Export? Access-to-C-subprogram? Both places where the C is doing the calling.]

Approve intent: 10-0-1. Niklas abstains.

At the end of the meeting, Steve asks about the semantics of **abort** Some_Task_Subtype_that_Includes_Null'(**null**). Does it raise Constraint_Error or do nothing? (Really! He quickly empties the room, er Zoom. He doesn't get an answer.)

AI22-0154-1/05 Revised resolution of indexing aspects

Gary: we discussed this last time, but more problems were found. Those were minor, the rest are handled in a new AI (AI22-0159-1). We look at that AI before voting this one. Ultimately, we voted both AIs together.

Approve AI: 9-0-1 (Jeff abstains). [Editor's note: While the actual vote was "with changes", the changes were only in AI22-0159-1; this AI was unchanged.]

AI22-0157-1/08 Fix three holes in definition of "known on entry"

Randy suggests changing the title to "four holes"; Steve suggests dropping the number.

Steve gives the example of ‘Constrained as something that we could consider but we didn’t. It can be applied to unconstrained **in out** parameters to determine a property of the actual. Tucker and Randy note that this seems valuable as it is similar to array bounds. If someone needs to test whether discriminants are changed by a subprogram, knowing whether the parameter is constrained seems to be part of the test. We decide to add it to the wording.

Steve tries Size as another such example. We agree that isn’t worth it. He also gives the example of a declare constant that is initialized with a known-on-entry expression. That also doesn’t seem worth it.

Approve AI with changes: 10-0-0.

AI22-0159-1/01 Inheritance for aspect allowed to denote multiples subprograms

Steve asks what happens if the example had the parameter to Index be A : Fine_String_Map’Class. That would be illegal by 13.1.1(18.2) rules; these rules are always rechecked for derived types (whether or not the aspect is given explicitly) for nonoverridable aspects.

Gary notes that he did try to implement this, and he thinks it is possible.

Approve AI with changes: 9-0-1 (Jeff abstains).

AI22-0160-1/01 Improved variable-vs-constant indexing

Steve explains the AI. Randy and Tucker help. The main issue is that the prefix of ‘Access needs to be aliased, and constant indexing is not necessarily aliased. In the example, if the constant indexing was used, the program would be illegal as the constant indexing is not aliased. A secondary issue is that an access-to-variable needs a variable indexing and the rules do not currently require that.

The fix is just to use the variable indexing (if it exists) for the prefix of ‘Access (and ‘Unchecked_Access); it is always aliased and will work no matter what kind of access is involved.

Tullio notes a typo in the summary.

A question was raised as to whether the “for the purposes of this rule” is acceptable. Tucker explains that we didn’t want to have to check the entire RM as to whether some global change would cause a problem in some other case. We think there are other instances of such rules in the RM (no one showed an example, however). [Editor’s note: When creating these minutes, I used the RM search tool to find occurrences of this phrase in E.2.1(7.4/5) and H.4.1(4/5). So there is existing wording using it, but all of the uses I found are fairly recent.]

Approve AI with changes: 10-0-0.

AI22-0161-1/02 Provide Unicode and universal Text interfaces

Tucker explains his AI, with a goal of a single type that is a universal string which can represent any Unicode string, and which can be used for general text usage.

Tucker focused just on defining the types, including types to convert to.

He has iterators so one can treat it somewhat like an array of character.

Randy notes that Wide_String and UTF-16 essentially are compatible. There doesn’t seem to be any need to treat them separately. In particular, the “surrogate” characters (that are used to represent code points with more than 16-bits) are not allowed in legitimate text. So any Wide_String that contains them is either UTF-16 or garbage. Tucker worries that some users are expecting a one-to-one mapping of characters to components. Robin notes that there is nothing in the definition that prevents surrogates, and there is a lot of stuff that can be done that makes nonsense (such as splitting a grapheme) that can’t reasonably be prevented with any rules about the contents.

Tucker defers to Robin's greater knowledge in this area (including what other languages do). Thus, the only remaining issue is whether we want a separate UTF-8 type.

Jean-Pierre says he originally wanted it to be a separate type, but he was convinced that one wants to read it as a String and then determine the encoding. Tucker thinks that might argue to make it a derived type. We don't seem to come to a conclusion on this point.

If this doesn't depend on AI22-0148-1, then we want some sort of discriminant. If it does, no discriminant is needed (bounds would come from the Bound aspect if desired). Tucker notes that he wanted a low overhead type, and specifically does not want the type to be controlled.

Justin suggests using the Locale to determine how characters are defined. He says most users only are using a single language. Robin lanuches into a fairly detailed description of how that is defined in Unicode. No one really followed it, but his point was that this is a very complicated mechanism in Unicode, and we don't have (or likely want) enough Unicode support to wade into those waters.

Robin notes Unicode tries to keep what is a grapheme stable, but mistakes are made.

Tucker asks if the Grapheme package is useful, Robin thinks it is. It definitely is not particularly hard to implement, but it will need changes from time to time as Unicode evolves. We should not tie the language definition to a specific Unicode version, as that will always be out of date. (Luckily, we already have a mechanism to avoid that.)

We turn to discussing Randy's alternative AI, AI22-0161-2/02. Afterwards, we decide to focus work on Randy's alternative and No Action this alternative to avoid confusion.

No action: 11-0-0.

AI22-0161-2/02 Universal Unicode Text

Randy explains his philosophy. He believes that we need a universal string that can be used to replace essentially all existing text manipulation other than interfacing to other things: legacy code, foreign code, files and streams with defined formats. In order to do that, we need a new string type that both handles all of the Unicode characters *and* can be manipulated efficiently. Tucker's version omitted that last part; there isn't any obvious way to do anything interesting with his Text type.

Randy expects that every Ada operation that takes or returns a string (built-in or in a library package) should get a new version that takes or returns a Text. It is the constant need to convert to/from String that makes Ada.Strings.Unbounded so painful to use. We have to avoid that here.

He included all of the operations from Ada.Strings.Unbounded (removing all of the redundant ones) to show how the package(s) would work in practice. He also included the character and grapheme iterators, some normalization routines so that untrusted inputs can be made consistent, and defined the attribute Text_Image (along with the needed support in the text buffers). More is needed but this seemed like enough to show the general idea.

Tucker would like to have minimal operations along with the basic type. He worries that in low overhead situations, dragging in a lot of unused subprograms could discourage some users from using a Text string. Randy wonders why those users don't get a compiler that doesn't include dead code (it seems fundamental to him, coming from super-low overhead requirements). Tucker says that we have to work with the existing situation (meaning GNAT does not easily eliminate unused subprograms from a package).

Randy says that things like searching could be in a child package. Tucker would be happier if that was the case. He thinks that the cursor is needed for slices, it hard to imagine doing anything without slicing or individual character reading.

Tucker mentions that he doesn't think this type should use finalization. Randy says that assuming that AI22-0148-1 is available, he expects type Text could be implemented with just a 32-bit length and an octet array. (Octets being an ISO and Unicode name for 8-bit bytes.) [Randy's note: I didn't mention it at the meeting, but my proposal *requires* Text to stream with this representation, regardless of what the actual representation is. That makes streaming portable between targets and between implementations, regardless of bounded or other representation issues.]

Indeed, the existing unbounded strings could be reimplemented this way if someone wanted to reduce the overhead. Bounding would be available using the Bound aspect. If AI22-0148-1 is not available, then we'd need an unbounded, controlled version, and a bounded, uncontrolled version with an octet Capacity discriminant.

He notes that he wrote a short example to show how these operations could be used. We briefly look at it. It is noted that we really want the Text_Value attribute; Randy will add that here.

Robin says that it is much easier to review this version than the alternative.

Robin notes that it is nice to stick closely to Unicode terminology. Randy notes that we don't want to stray too far from usual Ada terminology. He chose Unicode_Character as it is a character type, and he thought the name needed to reflect that. The definition of it clearly needs to tie to a Unicode scalar value (which he has done).

We talk about Text_IO. Tucker would prefer to use Forms to control the output format. Randy notes that we need versions of the string manipulation subprograms found in Integer_IO and Float_IO and so on – those have nothing to do with files. We start to feel bogged down on this topic and decide to defer that (it wouldn't be in this AI anyway).

There should be an "Nth" character to cursor function. It is essentially Add_Character(First, N-1), but that is clunky to write.

We agree that most operations should go into one or more child packages. Only the most basic operations should be in the main package. Randy wonders if Replace_Slice needs to be included in the "basic" subset. It is the only sensible way to modify a string. The group suggests that mutating a string is much less likely than reading from it. But one still needs to build up a result string somehow, which often means extracting something from one string and adding it to another. We certainly would need the Slice and concatenation operations for that.

Tucker isn't so sure. Randy thinks that the cursor operations are fundamental. One needs them to iterate the string, to slice the string, and pretty much to read anything from the string (outside of the single character indexing routine Element). Next/Previous/First/Last are needed for iteration, and are also useful for string decomposition. It in particular doesn't make sense to have a cursor for iteration and a separate cursor with essentially the same meaning in a child package for operations.

Approve intent of AI: 11-0-0.

Detailed Review of Ada Github Issues

#95 'Value for enumerations

The suggestion ultimately is 'Valid_Value, which returns True if the string passed would succeed when passed to 'Value. GNAT has this. Justin will take this one.

#111 size of immutably limited unconstrained discriminated components

This Issue is already closed, it shouldn't be on the list. Randy will try to take care of that.

#119 Unchecked_Deallocate_Subpool should be preelaborated.

Jeff notes that this was already done by AI22-0050-1. This one should just be marked as handled and closed. This was a duplicate issue, the original issue was #20.

#137 Max_Stream_Size for composite types?

Steve worries that this could be very large for unconstrained record types. Tucker notes that it is like Max_Size_in_Storage_Elements. Randy reads the AARM note for that attribute, which says it can be very large for unconstrained types. Sounds like it is expected.

Someone mentions that he has had to do this calculation in programs, and it is painful.

Tucker will take this one (it was his idea).

#156 Suggested enhancements to the wording in 3.10.2 about "tied" accessibility levels

This is assigned to Randy to add the suggested AARM notes.

#170 13.11.5 Subpool Reclamation: Is it the compiler's or user's responsibility to prevent loops?

Brad will look at this one.

#173 Illegal class-wide preconditions on Allocate/Deallocate_Subpool in 13.11.4

Brad will take this, possibly to combine with the other issue.