Minutes of the 29th ARG Meeting

18-20 November 2005

Atlanta, Georgia, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Gary Dismukes, Pascal Leroy, Steve Michell, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg (Friday to 3:30 pm only), Tucker Taft, Bill Thomas (except Sunday), Joyce Tokar (except Saturday morning).

Observers: Greg Gicca (Friday and Saturday morning only)

Meeting Summary

The meeting convened on 18 November 2005 at 09:10 hours and adjourned at 11:20 hours on 20 November 2005. The meeting was held in a conference room at the Doubletree Buckhead Hotel in Atlanta, Georgia, USA. The meeting covered the entire agenda as well as various off-agenda items.

AI Summary

The following AI was approved:

AI-436/01 Record representation clauses for limited record types (9-0-0)

The following AIs were approved with editorial changes:

AI-434/02 More presentation changes to the Standard (10-0-0)
AI-435/01 Storage pools for access-to-subprogram types (9-0-0)
AI-437/02 Glossary updates for the Amendment (10-0-1)
AI-438/01 Stream attribute availability for limited language-defined types (8-0-3)
AI-440/01 Index of language-defined entities (by acclamation)
AI-441/01 Passing a null stream access value to stream attributes (7-0-2)
AI-442/01 Classes and categories of types (5-0-3)

The following AI was voted no action:

AI-439/01 Transitioning to Ada 95 (9-0-2)

In addition to the AIs, numerous corrections to existing AIs were approved, organized by paragraph in the Draft 14 AARM. See the detailed minutes for more on these corrections.

Detailed Minutes

Meeting Minutes

There are no comments on the minutes.

The minutes of the 28th meeting are approved by acclamation.

Schedule of the Amendment

The corrections from this meeting should be applied by December 15th, probably earlier. WG 9 has decided on a review period until February 1st (assuming the drafts are available on time). The draft of the Amendment will be circulated (the AARM is totally unofficial for this purpose, although it is recommended that the AARM be pointed out to potential reviewers).

Pascal asks when the final reference manual formatting will occur. Once the Amendment is cast in concrete (in February), the RM will start being widely used. Erhard suggests that other than pagination, formatting can be done once this clears WG 9. Pagination should wait until after SC 22.

Erhard is quite concerned that we don't make any manual changes at this point. He also is worried about spending a lot of effort on tooling to avoid manual changes. Randy points out that much of the remaining effort is tooling (it doesn't make sense to repeatedly manually change the "this paragraph was deleted" marks, for instance, or the broken note numbers). Tooling work can (re)start on January 2 (that's when more money is available).

Should we try to get articles about Ada 2005 into various publications with wider visibility to the programming community? Yes, of course. Several people mention that they have or used to have contacts in various journals. They are asked to see if they can get editors interested and to determine the nature of the articles appropriate for each journal.

Date and Venue of the Next Meeting

Pascal suggests that we meet twice a year in the future, preferably co-located with the conferences and WG 9. So the next meeting would be in Porto, Portugal. June 9-11, 2006 (afternoon of Friday, all day Saturday and Sunday), co-located with the Ada Europe 2006 conference.

There is a slight chance that we'd need a meeting to handle troublesome comments from the WG 9 review. Randy tries to draft Joyce into hosting such a meeting, but we agree to postpone any consideration until such time as it becomes clear that we need to meet.

Document Formats

Randy says that we had concluded in e-mail that making the tools for creating the documents available is what is required, along with formats (text, HTML, PDF) primarily intended for reading but not modifications. We can't prevent modified versions, and significant changes can add value for users. But simple changes are not a good idea; they just make different versions with no significant benefit to users and a significant potential for confusion.

Pascal will research making digital signatures available for the documents, so that users can check that they have the original versions.

Support of Ada 2005 in Ada 95 compilers

WG 9 addressed this yesterday with a resolution asking the ACAA to spend its efforts on assessing conformance to Ada 2005.

Future Maintenance of the Ada Standard

Steve Michell reports that WG 14 and WG 21 create small Technical Reports for adding new functionality. TRs are more lightweight than a full standard. Such a TR would look like a secondary standard (not a like a delta as does the Amendment). Such reports could be promoted to an Amendment if appropriate.

Pascal says that this would work for totally new stuff (packages, pragmas, attributes), but not necessarily for new syntax. In any case, we should plan to produce a new Technical Corrigendum in roughly five years (there will be bugs to fix). As long as we continue to produce wording for each new AI, this should not be a major task.

Technical Reports remain an option that we should consider, but we need to wait until we get feedback from the community to see what kind of evolution is more appropriate (we are talking 1-2 years to start getting valid experience reports).

Revision of the ASIS Standard

WG 9 instructions emphasize adding Ada 2005 features to ASIS, and keeping compatibility as much as possible. Erhard notes that there is an opportunity to add layers or new functionality to make it easier to use.

Pascal thinks that this would dilute the effort; we need to get Ada 2005 features in ASIS soon. Ed asks what's missing. Jean-Pierre says that there is missing code for dispatching calls; you can't get to the declaration. Other ASIS users also have missing features.

Pascal says that we have short-term needs to address, but he's not against future work to extend it. Tucker agrees with that, but says that we need new features in order to get people's energy level increased. People are unlikely to be excited about adding **limited with** to ASIS.

The role of the SIGAda working group (ASISWG) is not yet clear. But acting as a filter for ideas before bringing them to the ARG seems useful. That worked well for the real-time group. Should they have workshops? Pascal says that he doesn't want to constrain how the ASISWG works.

Jean-Pierre asks what the output of this process is? We haven't actually decided that yet. Are we happy with the format of the existing standard? Hardly anyone has looked at it recently. Most of the information is in comments to the specifications of the packages. There is a special permission waiving copyright on the package specifications.

Tucker notes that it means that a revision would be not a problem in that case; no one uses the section numbers anyway. And an Amendment would be double the size, as it would be "replace this package with that package". A revision would show good will toward ITTF as well, for little cost.

Erhard points out that we need to be careful that the specifications are freely available. Someone says that the standard would be useless otherwise; but ISO never has had a problem making useless standards.

Pascal will take an action item to discuss with Jim Moore the availability of the specifications.

Bill and Greg will collect technical issues for rough drafts.

Randy will look into extending the tools to handle new AIs. He will need to do this for Ada 2005 anyway; we can't continue to use "AI95" as the name of these AIs. This triggers a short discussion, which concludes that starting the numbers over at 1 is preferable. But the file names need to remain distinct; in order to do that we will use longer file names for Ada 2005 AIs: AI05-00001, etc. For ASIS, we'll use the prefix SI99 (as the standard is 15291:1999).

Thanks

The ARG thanks our host, SIGAda for the accommodations for the meeting.

Old Action Items

Randy finished the AARM and Amendment documents on October 30th, too late for a letter ballot.

Bob Duff reviewed the entire core of the AARM, but then ran out of energy.

All other reviews and action items were completed by their deadlines.

New Action Items

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

Randy Brukardt:

- Finish draft 15 of the Amendment and a matching draft of the AARM by December 15th.

Editorial changes only:

- AI-434
- AI-435
- AI-437
- AI-438
- AI-440
- AI-441
- AI-442

Greg Gicca:

- Collect ideas for ASIS improvements; write SIs to present the ideas to the ARG (with Bill Thomas).

Pascal Leroy:

- Investigate digital signatures of documents.
- Discuss with Jim Moore the requirements for reasonable availability of the package specifications for ASIS.

Bill Thomas:

- Collect ideas for ASIS improvements; write SIs to present the ideas to the ARG (with Greg Gicca).

Detailed Review

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

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Draft 14 Consolidated AARM. Paragraph numbers in other drafts may vary.

In some cases the Editor has uncovered problems in the wording agreed upon during the meeting, and has corrected that wording. When that happened, the discussion of the problem and the final wording are shown in bold **like this**.

Detailed Review of Amendment AIs

AI-437/02 Glossary updates for the Amendment

John and Randy created this set of changes and new definitions. There are many other terms that could have been included (like formal parameter, actual parameter, instance, library level, and rendezvous), but we have to stop somewhere.

Pascal notes that the key here is to never lie; it is OK to be vague.

We review the changes by paragraph.

N(1/2) Tucker suggests:

"This Annex contains informal descriptions of some {of the} terms used in this International Standard. The index provides references to more formal definitions {of all of the terms used in this International Standard}."

N(1.1/2) Tucker suggests "which" to "that". John says "but which" is better than "but that". No change is made here.

N(3.1/2) This is wrong (derived type is not universal, it doesn't cover interfaces, for instance). "An ancestor of a type is the type itself or, in the case of a type derived from other types, its parent type or one of its progenitor types or one of their ancestors."

N(4.1/2) There is a lengthy discussion on retaining "also". Finally, we take a straw poll on retaining "also": 4-2-2. No change here.

N(6) Add "(of types)" after class, to match category.

N(13/2) Steve Michell asks that the second sentence start with "The first type given in the derived type definition". Others say it is too pedantic. He then suggests "The first of those types", which is agreed to be lighter.

N(13.1/2) Tucker suggests: "A type is a descendant of itself, its parent type, its progenitor types, and their ancestors." Or, better, "A type is a descendant of itself, its parent and progenitor types, and their ancestors." Steve Baird suggests adding "If type T1 is an ancestor of T2, then type T2 is a descendant of T1". No, this is not a place for mathematical definitions. Tucker suggests: "Note that descendant and ancestor are inverse relationships."

So we agree on:

"A type is a descendant of itself, its parent and progenitor types, and their ancestors. Note that descendant and ancestor are inverse relationships."

Add the last sentence to the definition of "ancestor".

N(15/2) "[of]{for} a composite type" in the first sentence, and similarly in the third sentence. Drop type: "…if the component [type] is an array [type]."

N(15.1/2) No changes.

N(17.1/2) No changes.

N(19.1/2) "is used" should be "can be used". Later (see N(30.1/2)), we change to "can be called".

N(20.1/2) "incompete" (only in the AI). Change "are valuable" to "can be used" (also in "private type").

N(21.1/2) "a derived type" should be "another type" thus: "An interface type is a form of abstract tagged type which has no components or concrete operations except possibly null procedures. Interface types are used for composing other interfaces and tagged types and thereby provide multiple inheritance. Only an interface type can be used as a progenitor of another type".

N(23/2) "such as for an assignment_statement" is weird. "such as in an assignment_statement". Drop both "(a view of)".

N(24.1/2) Add "primitive" between "inherited" and "operation".

N(25.1/2) Tucker suggests "almost any kind of type"; "mentioned" should be "given".

Erhard asks that this be "parent type". But this defines a relationship, not a kind of type. We'd also have to change "ancestor", "descendant", and "progenitor" if we change this one. Since we're not going to change this, drop "type": "The parent [type] of the…". "The parent [type] can…".

So we end up with: "The parent of a derived type is the first type given in the definition of the derived type. The parent can be almost any kind of type, including an interface type".

N(29.1/2) Tucker wants "properties" to be "components". Randy argues that other properties are hidden, such as primitive operations. Pascal says that the completion could be a task; surely that's not a "component", and it's hidden. Straw vote: "component" 2; "properties" 6; abstain 1. So no change here.

N(30/2) The AARM has "which hide" instead of "that hide". Another straw poll: "which" 2; "that" 5; abstain 3. "detail" should be "details". Also previously noted change (see N(20.1/2)). Thus: "A private type gives a view of a type that reveals only some of its properties. The remaining properties are provided by the full view given elsewhere. Private types can be used for defining abstractions that hide unnecessary detail{s} from a client".

N(30.1/2) Drop "stand alone"; "can only be called as a statement".

Steve Baird would like parallel wording for functions [N(19.1/2)]. "and can be called as part of an expression."

So we end up with: "A procedure is a form of subprogram that does not return a result and can only be called as a statement".

N(30.2/2) "A progenitor [type]…" (two places). "mentioned" should be "given".

"derived type" is wrong here (neither interfaces nor tasks are derived types). Add as the last sentence "Interfaces, tasks, and protected types may also have progenitors".

So the final wording is: "A progenitor of a derived type is one of the types given in the definition of the derived type other than the first. A progenitor is always an interface type. Interface, task and protected types may also have progenitors".

N(33/2) This is still confusing. Use: "A protected type is a composite type whose components are accessible only through one of its protected operations which synchronize concurrent access by multiple tasks".

N(37.1/2) No changes.

N(38/2) "and/or" is ugly. Pascal suggests just "or" here. It is highly unlikely that both will be used on a single subtype, and this isn't necessarily exclusive or anyway. Thus: "A subtype is a type together with a constraint or null exclusion, which constrains the values of the subtype to satisfy a certain condition. The values of a subtype are a subset of the values of its type".

N(38.1/2) "can be used with" should be "can be an ancestor". Better, replace the last sentence with: "A synchronized interface can be an ancestor of a task or a protected type. Such a task or protected type is called a synchronized tagged type".

N(40) This needs to be worded to be similar to protected types. "A task type is a composite type used to represent active entities which execute concurrently and which can communicate via queued task entries. The top-level task…"

N(42/2) "A view of an entity reveals some or all of the properties of the entity. A single entity may have multiple views."

Randy asks whether any of these changes should change the normative wording in cases where they are shared. The answer is no, they should be separated if necessary.

Approve AI with changes: 10-0-1.

AI-440/01 Index of language-defined entities

Tucker would prefer to drop "and Values" from Q.5. The group agrees.

It's noted that the list in Q.5 does not include "named numbers" (they're not constants by the definition of 3.3). These should be added.

Approve AI with change (by acclamation).

AI-441/01 Passing a null stream access value to stream attributes

Randy explains the problem, and the amazing fact that the change is *not* incompatible.

Not everyone understands the latter point; Randy explains that a subprogram without a null exclusion on the stream parameter could be used to specify a stream attribute, because only *mode* compatibility is required (which ignores constraints and exclusions). There is some discomfort with this; couldn't this cause problems? Randy says no; there is a separate legality rule for the parameter of the type; so only the stream attribute parameter has freedom. And the worst that could happen is that checks are made both on the call (because the attribute is null excluding) and inside of the subprogram (because the subprogram was not null excluding).

Gary says that a direct call might change where an exception is handled. True, but this is caused by the fact that access parameters are not null excluding, not by this change (which actually causes less change from Ada 95).

Pascal thinks that allowing **not null** in Ada95 compilers is now unnecessary, since adding **not null** to the stream attributes is compatible. (We didn't realize that when the AI was drafted.) Tucker says that the semantics *will* change when you move to Ada 2005; if you can put these in now, you can avoid such a change.

Pascal doesn't think this issue is important enough for such a permission. The market should decide if there is a need for a transition mode in compilers.

So we drop the Ada 95 part of the AI.

Approve AI with change: 7-0-2.

AI-442/01 Classes and categories of types

This AI came out of attempts to fix a number of problems with the wording of the Standard. Pascal, Randy, and Tucker (or was it Huey, Dewey, and Louie?) discussed this extensively and eventually came up with this AI.

Erhard worries about such a major complication. Pascal, Randy and Tucker all argue that this is necessary and fixes a number of holes and places where the Standard was fudging.

In 3.2(13/2) "classification" should be "categorization".

Steve Michell points out that the text 3.4(1.1/2) is a new paragraph, and needs to be described as such in the AI.

Add "for example" in front of the list of possible categories. "of a type in the set" instead of "such a type". Better change the last "class", too, giving: "By saying that a particular set of types forms a class, we are saying that all derivatives of a type in the set inherit the characteristics that define that set".

3.4(8/2) Steve Michell suggests making this a bulleted list. But that would be very long, it would have to have two columns, and we don't have a way to do that currently. We could make it into a table of some sort, but we decide to make no change.

Does this need to mention any other kinds of class (such as derivation classes)? No, this bullet defines the characteristics of the derived type; there is no need to mention other kinds of class here. Those other characteristics are inherited by other bullets here.

Steve Baird argues that this is view-specific, which would cause problems. Pascal says that the *type* determines the characteristics, even though you can't necessarily see those characteristics (they might be hidden). Pascal says that the existence of private types would mean that nothing was a class, as no types would be closed under derivation.

Steve Baird asks why a private type completed by an integer type doesn't match a formal integer type; Pascal just told us that the category is determined by the underlying type. And 12.5(7/2) says that only the category is relevant for matching.

Pascal suggests that we say that the actual type must have the characteristics of that category.

Tucker says that we have to be able to talk about categories of views. It's common in the standard to omit the word view where it might be needed formally. Moreover, there is no problem with private types; they form a new derivation. Indeed, private extensions are the reason that tagged records do not form a class. So this was already taken into account.

So 12.5(7/2) is talking about the current view (similar to many other rules in the Standard), and it is fine as is. In any case, this is no different than the original wording, which would have the same problem if views were not taken into account.

Approve AI with changes: 5-0-3.

Erhard is still worried that this is not important enough for such a significant last minute change. Pascal argues that that would have been true in Ada 95, but this is much more broken in Ada 2005 — we have many more things that aren't classes (but nevertheless were called classes in the wording). No one's mind is changed.

Detailed Review of Regular AIs

AI-434/02 More presentation changes to the Standard

Question (11) is confused; the rule is "not in C.7.1". The answer says "andsince", which should be two words.

Question (31) has an extra comma.

Erhard is worried about the contents (not the wording) of the question (6) note - C.3.1(23/2). He will research the problem, and report to us if there is any problem.

Tucker objects to question (26). He notes that 3.3.1(10) says that subtypes have implicit initial values, not objects. Therefore, this change is wrong. Pascal would like to have different values for different objects of a subtype. We agree to change to "implicit initial value{s}" (adding the 's') and drop the "object of" wording (for both paragraphs).

In Question (30), Tucker called 13.13.2(8.1/2) and 13.13.2(25/2) "inscrutable". He provided rewrites.

Erhard says that inherited operations are "specified" by 13.1(17-18); they're *not* default implementations. We could drop the word "default", but that's uncomfortable, as it sounds like they're not allowed to be specified.

Pascal suggests we drop inheritance altogether, and just have those attributes call the parent.

Tucker suggests "For untagged derived types, the default implementations of Write and Read are defined to invoke the corresponding attribute of the parent type; …" We'll use similar wording for 13.13.2(25/2).

Erhard worries that this is an unnecessary change. Pascal argues that this makes the stream attributes better defined, because inheritance is not very well explained in 13.1 (it is missing the type conversion stuff that is defined for subprograms). We should add an AARM note to explain this.

Approve AI with changes: 10-0-0.

After the meeting, we discovered that this wording is incompatible with Ada 95. Therefore, the wording was dropped from the AI and not included in the next draft (Draft 15); rather the Draft 14 wording was retained, and a separate AI (AI-444) was prepared on the topic.

AI-435/01 Storage pools for access-to-subprogram types

It doesn't make sense to talk about the storage pool of an access-to-subprogram type.

Correct the spelling of "objct" in the question.

Approve AI with change: 9-0-0.

AI-436/01 Record representation clauses for limited record types

No one has any idea why the incompatible change banning record representation clauses from limited types was made in Ada 95.

Approve AI: 9-0-0.

AI-438/01 Stream attribute availability for limited language-defined types

Correct the spelling of "geneator" in the discussion of the AI.

Tucker says he doesn't like the nested package (despite the fact that it was originally his suggestion!). He suggests an implementation-defined package including a type that has the appropriate operations. Then Exeception_Occurrence can be derived from this type. This seems like too much mechanism, and Tucker withdraws his suggestion.

Gary doesn't believe that making these primitive is a problem, especially if the names are unusual enough. Deriving from Exception_Occurrence is rare. Tucker suggests Read_Exception_Occurrence and Write_Exception_Occurrence. Someone suggests that using Greek letters we could avoid any incompatibility. Steve Baird says that Rho would work, and then you could rho, rho, rho your occurrence. General laughter ensues. In any case, this seems too goofy.

We will drop the nested package, and use the names that Tucker suggested.

Approve AI with changes: 8-0-3.

AI-439/01 Transitioning to Ada 95

In the AI question, Wide_String should be Wide_Wide_Character.

Pascal objects to an open permission to add things, because such additions could be incompatible. He would prefer to have a list of allowed additions, because they could compromise portability. A user could accidentally use a routine that would be defined only in some compilers.

There is a sufficient flexibility in the language already to handle this (using a non-standard mode, perhaps). GNAT uses a pragma to flag new items, they give errors in Ada 95 mode. So we are not going to give any permission in this sense.

We will listen to compelling cases for including such things, but no one can think of anything that might qualify.

No action: 9-0-2.

Corrections to existing AIs

The corrections are listed by the paragraph numbers in Draft 14 of the AARM, followed by the AI and version to which they apply. They were not necessarily discussed in the order in which they are presented. The first paragraph usually describes the proposed correction; other paragraphs the discussion (if any), and the last paragraph the vote on the correction.

Many of the corrections to Draft 13 of the AARM were voted as a block; these are indicated by [Draft13] after the vote. Similarly, many of the corrections to Draft 14 of the AARM were voted as a block; these are indicated by [Draft14] after the vote. Votes not so marked were taken individually.

Title

Jim Moore had found out that the ISO title for a consolidated standard would be

ISO/IEC 8652:200y Ed. 3

so we should use that in the page headers and on the title page (with "y" replaced by the appropriate year, either "6" or "7", once we know it).

Approve correction: 11-0-0. [Draft13]

Foreword to this version (0.6/2)

Changed "not expected that" to "not known whether" to be more neutral on ISO's possible actions.

Approve correction: 9-0-1. [Draft14]

Foreword to this version (0.7/2)

One of the sentences mentioned only the Technical Corrigendum without the Amendment.

Approve correction: 11-0-0. [Draft13]

Introduction (38.1/2) AI-387/05

There is no such thing as a "synchronized type", so that was removed from this paragraph.

Approve correction: 11-0-0. [Draft13]

2.1(1/2) AI-395/10

The definition of "character" made Wide_Wide_Character a holey enumeration type (as FFFE and FFFF were not considered characters). This also changes 2.1(3.1/2).

Approve correction: 11-0-0. [Draft13]

2.1(3.1/2) AI-395/10

See previous item.

Approve correction: 11-0-0. [Draft13]

2.1(10/2) AI-285/17

Unicode 4.0 uses "Number, Decimal" rather than "Number, Decimal Digit"; the wording was changed accordingly.

Approve correction: 11-0-0. [Draft13]

2.1(13/2) AI-285/17

This was very ugly; after further research, it was determined that a much simpler version would suffice.

Approve correction: 11-0-0. [Draft13]

2.1(14/2) AI-395/11

Change "which" to "that".

Approve correction: 9-0-1. [Draft14]

2.3(3.1/2) AI-395/10

Another occurrence of "Number, Decimal Digit".

Approve correction: 11-0-0. [Draft13]

3.2(12/2) AI-345/12

Since "interface" is not a class, it shouldn't be shown in a hierarchical chart as a stand-alone item.

This was discussed in the context of AI-442; that AI was approved 5-0-3.

3.2.3(6.1/2) AI-335/04

Clarified the wording to ensure that it applies to package specifications as well, by using the new term "declaration list", introduced by a correction to AI-420. Randy explains that the term "declaration list" was created to avoid issues with "list of declarative_items", as a package specification does not have declarative_items.

Gary wonders if this rule is wrong. Consider a limited type in the visible part, and the stream attributes defined in the private part. These are not primitive. Yes, that's intended, because there is no guarantee that the descendants have these operations. That would be a disaster for dispatching, we'd be potentially executing non-existent bodies.

Tucker doesn't want to make stream attributes primitive, that seems to imply inheritance even through attributes are not inherited. He proposes that we add this to 3.9.2(1):

"The primitive subprograms of a tagged type, the subprograms declared by formal_abstract_subprogram_declarations, and the stream attributes of a specific tagged type that are available (see 13.13.2) at the end of the declaration list where the type is declared are called *dispatching operations*."

Once this is added, we need to drop 3.2.3(6.1). The group agrees that this seems better.

Approve correction with changes: 11-0-0. [Draft13]

3.2.3(7/2) AI-200/04

"In the case of" was changed to "For" to be consistent with the preceding bullets.

Approve correction: 9-0-1. [Draft14]

3.3.1(8.1/2) AI-373/08

Changed "which" to "that".

Approve correction: 11-0-0. [Draft13]

3.4(35.1/2) AI-401/05

Changed "which" to "that".

Approve correction: 11-0-0. [Draft13]

3.4.1(3/2) AI-230/19

This needed to be updated to match 3.2.1(8/2). Since 3.2.1(8) defines the terms, this was reworded as non-specifically as possible.

Approve correction: 11-0-0. [Draft13]

3.9(2/2) AI-345/11

The text "type(s)" was changed to "types".

This paragraph needs to mention synchronized tagged types, as it is supposed to be a complete list of the kinds of tagged types. Change the second sentence to: "In addition, an interface is a tagged type, as is a task or protected type derived from an interface (see 3.9.4)".

Approve correction with changes: 11-0-0. [Draft13]

3.9(2.1/2) AI-345/12

Added “non-interface” to the wording to make it clear that interfaces aren’t type extensions. Added “of some other tagged type” after “private extension”.

Tucker doesn’t think 3.9(2.1) reads like a definition. Drop “either”. Tucker says that makes it less clear that this is exhaustive. Ed suggests “one of the following”. Tucker adds a “colon”. Gary says we need to repeat “non-interface”.

We should just use “synchronized tagged type” here. So: “... or a non-interface synchronized tagged type (see 3.9.4).”

The extra “of some other tagged type” seems redundant. Replace the first sentence with:

“Every type extension is also a tagged type, and is a *record extension* or a *private extension* of some other tagged type, or a non-interface synchronized tagged type (see 3.9.4).”

Approve correction with changes: 9-0-1. [Draft14]

3.9(6/2) AI-362/07

Randy had an action item to look at all of the packages that were recategorized to preelaborable, and see if any had private types that should have Preelaborable_Initialization. Only Tag qualifies and seems like it should have Preelaborable_Initialization. Thus, the pragma was added to type Tag.

Tucker jokingly suggests that Preelaborable_Initialization should be called P26N, much like I18N and L10N. Unfortunate that he didn’t have this idea earlier ☺.

Approve correction: 9-0-1. [Draft14]

3.9(18.2/2) AI-260-2/07

Aligned the colons of the parameters.

Approve correction: 11-0-0. [Draft13]

3.9(25.2/2) AI-260-2/07

The text “the tag The_Tag” is silly. While “the The_Tag tag” would be consistent with the rest of the clause, that is just as silly, so “the tag” was dropped.

Approve correction: 11-0-0. [Draft13]

3.9(25.2/2) AI-260-2/08

Steve Baird says that this wording does not prevent lifetime issues for the created objects. He proposes to modify the wording as follows:

“An instance of Tags.Generic_Dispatching_Constructor raises Tag_Error if The_Tag does not represent a concrete descendant of T{ that has the same accessibility level as T}.”

Steve explains why this is needed. Presume a tagged type T, two tasks, and a global variable X of type Tags.Tag. Assume one of the tasks declares an extension of the tagged type T, and saves its tag into X. The second task reads the variable and uses Generic_Dispatching_Constructor to create an object. The accessibility levels of the two tasks are the same, but they are incomparable (they have different lifetimes), and the object could outlive the type.

Pascal and Gary think that this is far too conservative. Randy says this is the same rule as used for stream attributes; why does this need to be different? The answer is that the function for a stream attribute is declared at the place of the type; in this case, the instance could be in the right place (that is more nested than the type).

We agree that this needs some dynamically enclosing check. Steve Baird and Tucker will try to fix this off-line.

On Sunday, Tucker presents the result of their work:

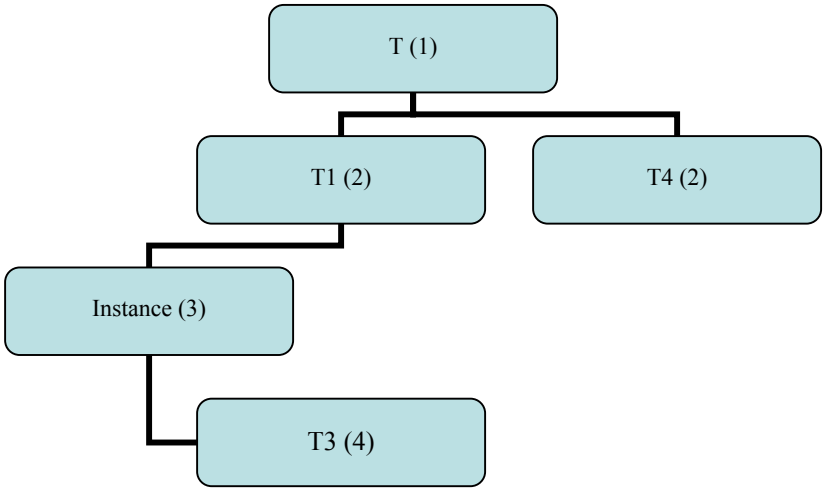
"…does not represent a concrete descendant of T whose innermost master is also a master of the instance."

This wording is a long negative clause; it's easy to lose the "not" here.

"…does not represent a concrete descendant of T or if the innermost master of this descendant is not also a master of the instance."

Gary asks about the impact on implementations. Randy wonders if we have to carry master information in the tag.

Tucker draws a bunch of balloons representing masters and their relationships (the accessibility level is in parentheses):



Tucker claims that you statically know the masters of the Instance and T. You have to be able to find out the masters of the type that the tag represents, so it does imply that the master is stored in the dynamic part of the tag.

An alternative is to use an indication of the stack frame that the type belongs to. Randy worries that this requires masters where none would otherwise needed. Tucker says that finalization is needed for leaving of nested tagged types anyway; the external tag has to be removed from a map.

We need to consider whether this should apply to Internal_Tag as well. Getting a tag for some other task isn't very useful. Tucker suggests adding some wording to 3.9(26/2):

"The implementation of Internal_Tag and Descendant_Tag may raise Tag_Error if no specific type corresponding to the string External passed as a parameter exists in the partition at the time the function is called, or if there is no such type whose innermost master is a master of the point of the function call."

But this says "may". So it still can happen. Tucker would like to make this a requirement; it would only be on Internal_Tag, as Descendant_Tag has a stronger check of its own. Randy and Pascal point out that using your own map of names (or something else) to tags is more likely. Making this a requirement also would require detecting types that don't exist (which we previously agreed not to require).

Tucker then suggests that we make this Implementation Advice. We need to define what this *should* return; that's more useful than saying what it should not return anyway. So add the following IA:

"Internal_Tag should return the tag of a type whose innermost master is the master of the point of the function call."

Note that the wording change to paragraph 26 is still necessary; we want the implementation to be able to raise Tag_Error if no appropriate type is available. But without the wording change to paragraph 26, if any type existed, even in the wrong task, Internal_Tag would be obligated to return it.

Approve changes: 9-0-1.

3.9(25.3/2) AI-260-2/08

Steve Baird is concerned that this paragraph only covers Generic_Dispatching_Constructor, and not the operations in Ada.Tags. Does the implementation have to be able to handle "bad" tags, either generating a "correct" answer or raising Tag_Error?? No, of course not! He is assigned some overnight work to produce wording to correct this.

On Saturday, Steve presents the result of his work. Replace 3.9(25.3/2) with

"If a value of type Ada.Tags.Tag identifies either

- a type that is not library-level and whose tag has not yet been created (see 13.14); or
- a type that does not currently exist in the partition (e.g., a type whose declaration was elaborated as part of an execution of a subprogram_body which has been left (see 7.6.1))

then execution is erroneous if this value is passed as a parameter in a call to any subprogram declared in package Tags, or to an instance of Tags.Generic_Dispatching_Constructor."

The (e.g. etc.) should be an AARM note, not in the normative text.

Pascal would like to revert to the original form:

"If {an}[the] internal tag provided to an instance of Tags.Generic_Dispatching_Constructor {or to any subprogram declared in package Tags} identifies {either} a type that is not library-level and whose tag has not been created (see 13.14), or {a type that} does not exist in the partition at the time of the call, {then} execution is erroneous."

Approve revised wording: 9-0-1.

3.9(26/2) AI-260-2/08

Most of this paragraph is made meaningless by the improvement to 3.9(25.3/2). The behavior is erroneous; that certainly includes Tag_Error as a possible result. So this means nothing for internal tags. But we still need it to cover external tags.

So, drop "tag or", and either delete or move the AARM note. Tucker doesn't like this wording.

"The implementation of Internal_Tag and Descendant_Tag may raise Tag_Error if no specific type corresponding to the string External passed as a parameter exists in the partition at the time the function is called."

Approve revised wording: 9-0-1.

Also see the discussion of 3.9(25.2/2), which makes the final wording:

"The implementation of Internal_Tag and Descendant_Tag may raise Tag_Error if no specific type corresponding to the string External passed as a parameter exists in the partition at the time the function is called, or if there is no such type whose innermost master is a master of the point of the function call."

3.9.1(1) AI-345/12

This paragraph was inconsistent with 3.9(2.1). It was changed to: "Every type extension is a tagged type, and is either a record extension or a private extension of some other tagged type, or a non-interface task or protected type derived from an interface type".

Drop “either” and change to “a non-interface synchronized tagged type.” (as we did for the matching paragraph)

Approve correction with changes: 9-0-1. [Draft14]

3.9.3(3/2) AI-260-2/07

One “declared by” was dropped, as it appeared twice unnecessarily.

Approve correction: 11-0-0. [Draft13]

3.9.4(5/2) AI-345/11

After much consideration, we agreed that task interfaces ought to be task types (similarly for other kinds of interfaces). Several paragraphs were changed. After this change, there is no need for special rules about task or protected objects.

Approve correction: 11-0-0. [Draft13]

3.9.4(6/2) AI-345/11

As part of making task interfaces be task types, this paragraph was split and wording added.

In the second paragraph (now 3.9.4(7)), Erhard would like to get rid of the parentheses, as they are ugly. Tucker suggests removing the “tagged” as well. So we are left with “an abstract”.

Approve correction with changes: 11-0-0. [Draft13]

3.9.4(12-16/2) AI-345/11

These rules were rewritten to be much more explicit about what is required. This was made necessary by making task interfaces a task type, although it’s an improvement in any case.

Approve correction: 11-0-0. [Draft13]

3.9.4(33/2) AI-433/05

The last sentence was rewritten.

Approve correction: 11-0-0. [Draft13]

3.10.1(9.1/2) AI-335/04

Clarified the wording to ensure that it applies to package specs as well (see discussion of 3.2.3(6.1/2)).

Approve correction: 11-0-0. [Draft13]

3.10.2(10/2) AI-416/13

Some cases were not covered by the list in this wording. The wording was rewritten to eliminate the list of syntactic categories.

There are more problems with this wording. The interaction between 7.6.1(3) and this paragraph causes some bizarre effects where parameters are finalized before the call of the function they are used in. For instance, the expressions in

a name would be masters when the name is used in a context other than a simple_statement. As an example, consider the following rename:

```
Obj : Some_Lim renames F(Lim => (others => <>)).C;
```

The parameter expression would be a master given the Draft 13 wording; that would mean that the parameter aggregate would be finalized *before* the call to F.

Randy, Pascal, and Tucker had met Thursday night in hopes of figuring out these issues. Afterwards, Tucker tried to figure out some principles and wording changes. He provided us with the following notes:

> What is the problem we are trying to solve? We should finalize temporaries and wait for tasks before we progress to the next statement, or begin a delay, or start a nested statement. On the other hand, we don't want to finalize a temporary while it is still being used. For anonymous allocators we also need to provide an accessibility level if they are the actual for an access parameter. The kinds of constructs that can represent temporaries that might need finalization are anonymous allocators, aggregates, function calls, and concatenations. Function results can be the prefix of a selected component, indexed component, or of some other kind of name.
>
> We need a master at every point where we want to require finalization or task waiting to occur, but not every temporary that is somehow "inside" a master need be associated with the innermost master. In particular, if a function_call or other construct itself has a value, it is unlikely it should be a master for this value.
>
> We need to know when you initialize an access discriminant with Blah'Access, whether it is legal — access discriminant of aggregate, subtype_indication of allocator, return object. We need to define the accessibility level of the access discriminants associated with a named constrained subtype.
>
> Shouldn't an aggregate have the same rule as a function_call?
>
> (See also 3.10.2(12/2) and 7.6.1(3).)

Tucker proposed rewriting this paragraph as:

- The accessibility level of an aggregate or the result of a function call (or equivalent use of an operator) that is used (in its entirety) to directly initialize part of an object is that of the object being initialized. In other contexts, the accessibility level of an aggregate or the result of a function call is that of the master immediately enclosing the aggregate or function call.

Tucker wonders if we could say expression instead. But then we're defining the accessibility level of "5" (and other values). That doesn't sound good. We stick with Tucker's original proposal.

Steve Baird wonders what "in its entirety" means: is this static or dynamic? That is, what about a slice that represents the entire object. That clearly isn't intended; this is something that is known at compile-time. No one is interested in adding more words to this to clarify. Perhaps we should add a "to be honest" note that would say "don't take any component or slice or the like, but parenthesis or qualification is fine". View conversions would be OK, value conversions would be not probably.

Tucker also had suggested reverting to the "enclosing" wording in 3.10.2(7). That just dredges up old arguments, and it is considered out of bounds.

Approve wording changes: 6-0-4.

One objection that we have had in the past (in the context of 3.10.2(7/2)) was that "enclosing" is a static concept but masters are a dynamic concept. So the wording incorporated into draft 15 is the following, which avoids "enclosing":

- **The accessibility level of an aggregate or the result of a function call (or equivalent use of an operator) that is used (in its entirety) to directly initialize part of an object is that of the object being initialized. In**

other contexts, the accessibility level of an aggregate or the result of a function call is that of the innermost master that evaluated the aggregate or function call.

3.10.2(12-12.3/2) AI-416/13

Tucker believes that these paragraphs are missing cases. He gave the following written notes; also see his notes in 3.10.2(10/2) for additional details and background.

> This is confusing, as it initially says it applies only to "an access discriminant in the subtype_indication or qualified_expression of an allocator, or in the expression or return_subtype_indication of a return statement." It then goes on to talk about access discriminant of an object defined by an allocator, and then "any other object with an unconstrained nominal subtype." This seems like apples and oranges.

He suggests revising these paragraphs as follows:

- The accessibility level of the anonymous access type of an access discriminant in the subtype_indication or qualified_expression of an allocator, or in the expression or return_subtype_indication of a return statement is determined as follows:
  - For an access discriminant whose value is determined by a discriminant_association in a subtype_indication, the accessibility level of the object or subprogram designated by the associated value (or library level if the value is null);

    Discussion: This deals with the following cases {when they occur in the context of an allocator or return statement}:

    - An extension_aggregate where the ancestor_part is a subtype_mark denoting a constrained subtype;
    - An uninitialized allocator where the subtype_indication defines a constrained subtype;
    - A discriminant of an object with a constrained nominal subtype, including constrained components, the result of calling a function with a constrained result subtype, the dereference of an access-to-constrained subtype, etc.
  - For an access discriminant of an object defined by an aggregate where the value of the discriminant is determined by a component_association in the aggregate, the accessibility level of the object or subprogram designated by the associated value (or library level if the value is null);
  - For an access discriminant of [any other object] {an object denoted by a name} with an unconstrained nominal subtype, the accessibility level of the object.
- The accessibility level of the anonymous access type of an access discriminant in any other context is that of the enclosing object.}

  Discussion: In other words, if you know the value of the discriminant from a discriminant constraint or an aggregate component association, then that determines the accessibility level; if you don't know it, then it is based on the object itself.

Tucker shows an example:

```
new T'(Obj)                -- accessibility comes from Obj
subtype S is T(D => Blah'Access);
new S;                     -- accessibility from first bullet
new T(D => Blah'Access)    -- accessibility from first bullet
new T'(D => Blah'Access)   -- accessibility from second bullet
Obj : T(D => Blah'Access); -- accessibility from new (outer) bullet
```

Pascal says that he has problems with the lead-in matching the bullets here. Tucker and Pascal reword the bullets (changes are marked below).

- The accessibility level ... is determined as follows:

- {If the value of the}[For an] access discriminant [whose value] is determined by a discriminant_association in a subtype_indication, the accessibility level of the object or subprogram designated by the associated value (or library level if the value is null);
- {If the value of the}[For an] access discriminant {is determined by a component_association in an aggregate} [of an object defined by an aggregate where the value of the discriminant is determined by a component_association in the aggregate], the accessibility level of the object or subprogram designated by the associated value (or library level if the value is null);
- {In other cases, where the value of the access discriminant is determined by an} [For an access discriminant of any other] object with an unconstrained nominal subtype, the accessibility level of the object.

- The accessibility level of the anonymous access type of an access discriminant in any other context is that of the enclosing object.}

Paragraph 12.2 needs an AARM note to say that this bullet is talking about an aggregate in the current context, while paragraph 12.1 is talking about any subtype_indication anywhere.

Approve wording changes: 6-0-4.

3.11(6.1/2) AI-420/03

Defined declaration list for use in other rules (see discussion of 3.2.3(6.1/2)).

Approve correction: 11-0-0. [Draft13]

4.5.2(9.1/2) AI-420/04

This rule was corrected to allow dispatching calls on user-defined "=", and to limit it to access-to-object types (access-to-subprogram types don't have designated types). An AARM note was added to explain why a similar exception is not needed for access-to-subprogram types.

It also was corrected to use "declaration list" (see the discussion of 3.2.3(6.1/2)).

Erhard suggests putting the "unless" first. That doesn't really work; Tucker suggests looking at this off-line. Tucker and Erhard will do this over lunch.

Later in the meeting, Tucker shows a proposed rewording for this paragraph.

"At least one of the operands of an equality operator for *universal_access* shall be of a specific anonymous access type. Neither operand shall be of an access-to-object type whose designated type is D or D'Class where D has a user-defined primitive equality operator that

- has result type Boolean,
- is declared immediately within the same declaration list as D, and
- has at least one operand that is an access parameter with designated type D or D'Class,

unless the predefined universal operator is identified using an expanded name with prefix denoting the package Standard."

This is ugly, as we have half of a sentence after the bullets. We need to put the "unless" part first.

"At least one of the operands of an equality operator for *universal_access* shall be of a specific anonymous access type. If the predefined universal operator is identified using an expanded name with prefix denoting the package Standard, the operands may be of any access type. Neither operand shall be of an access-to-object type whose designated type is D or D'Class where D has a user-defined primitive equality operator that

- has result type Boolean,
- is declared immediately within the same declaration list as D, and
- has at least one operand that is an access parameter with designated type D or D'Class,"

This isn't liked much either. Perhaps we should drop the bullets:

"At least one of the operands of an equality operator for *universal_access* shall be of a specific anonymous access type. Unless the predefined equality operator is identified using an expanded name with prefix denoting the package Standard, neither operand is of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator with result type Boolean, that is declared immediately within the same declaration list as D, and at least one of whose operands is an access parameter also with designated type D."

Tucker again comes to the rescue and suggests:

"At least one of the operands of an equality operator for *universal_access* shall be of a specific anonymous access type. Unless the predefined equality operator is identified using an expanded name with prefix denoting the package Standard, neither operand shall be of an access-to-object type whose designated type is D or D'Class, where D has a user-defined primitive equality operator such that:

- its result type is Boolean;
- it is declared immediately within the same declaration list as D; and
- at least one of its operands is an access parameter with designated type D."

We think this is good.

Approve wording: 8-0-2.

4.5.2(9.2/2) AI-230/19

Access type conversions often are allowed only one way, and we want an object of any type that is convertible to another type to be comparable to an object of that other type. We went a bit further and considered the predefined equals to be similar to

```
function "=" (Left, Right : access D) return Boolean;
```

Anything (as long as one is anonymous) that could call this routine should be allowed, especially as a user could write something like this. This eliminates accessibility from the equation, which convertibility brings in otherwise. The rules were changed accordingly, and the AARM note was replaced.

"shall be of access-to-…" in all cases, including the bullets, we're talking about the types of the objects.

Null should be bold; this is syntactic, not the value of the expression. What about (**null**)? Erhard suggests that "**null**" should be "of type universal_access". (That covers (**null**) as well.)

Approve correction with changes: 11-0-0. [Draft13]

4.5.5(19.1/2) AI-420/04

This rule was corrected to use "declaration list" (see the discussion of 3.2.3(6.1/2)).

This wording was modified as was the similar rule whose discussion was recorded in 4.5.2(9.1/2).

Tucker originally suggested:

"An explicit conversion is required on the result when using the above fixed-fixed multiplication operator when either operand is of a type having a user-defined primitive multiplication operator that

- is declared immediately within the same declaration list as the type, and
- has both formal parameters of a fixed-point type,

unless the predefined universal operator is identified using an expanded name with prefix denoting the package Standard."

This is ugly, as we have half of a sentence after the bullets.

"Unless the predefined universal operator is identified using an expanded name with prefix denoting the package Standard, an explicit conversion is required on the result when using the above fixed-fixed multiplication operator if either operand is of a type having a user-defined primitive multiplication operator such that:

- it is declared immediately within the same declaration list as the type; and
- both of its formal parameters are of a fixed-point type.

A corresponding requirement applies to the universal fixed-fixed division operator."

Approve wording: 8-0-2.

4.6(24.18/2) AI-230/19

Added a missing rule, so that **null** can be converted to a pool-specific type. Without this rule, **null** could be converted to a general access type, but not a pool-specific one.

Approve correction: 11-0-0. [Draft13]

4.9.1(1.4/2) AI-311/04

The second "both" was dropped to be consistent with 4.9.1(1.3/2).

Approve correction: 9-0-1. [Draft14]

7.1(6) AI-420/03

Defined declaration list for use in other rules.

Approve correction: 11-0-0. [Draft13]

7.3(20) AI-401/06

This note is wrong in the face of interfaces. Consider:

```
      type T is new I1 and I2 with private; -- I1, I2 are interfaces.
private
      type T is new Parent and I1 and I2 with ...
```

The ancestor type here is I1, and it has no relationship with the parent type (Parent).

The note was replaced by:

"The ancestor type specified in a private_extension_declaration and the parent type specified in the corresponding declaration of a record extension given in the private part need not be the same. If the ancestor type is not an interface type, the parent type of the full view can be any descendant of the ancestor type. In this case, for a primitive subprogram that is inherited from the ancestor type and not overridden, the formal parameter names and default expressions (if any) come from the corresponding primitive subprogram of the specified ancestor type, while the body comes from the corresponding primitive subprogram of the parent type of the full view. See 3.9.2.

Similarly, if the ancestor type is an interface type, the parent type can be anything so long as the full view is a descendant of the ancestor type."

The group doesn't like the single sentence added paragraph. This can be added to the next note (it's related anyway). So add the new paragraph to the beginning of note 8 (and drop "Similarly, ") "…given in the private part {also} need not be the same — the only requirement is that the private extension and the record extension be descended from the same set of interfaces."

Approve correction with changes: 9-0-1. [Draft14]

After the meeting is was noted that the introductory text to the second note talked about an ancestor type out of context, which is not good because notes should be understandable in isolation, so words were added to give the necessary context:

"If the ancestor type specified in a private_extension_declaration is an interface type, the parent type can be anything so long as the full view is a descendant of the ancestor type. The progenitor types specified in a private_extension_declaration and the progenitor types specified in the corresponding declaration of a record extension given in the private part also need not be the same — the only requirement is that the private extension and the record extension be descended from the same set of interfaces."

7.6.1(3/2) AI-416/13

This paragraph has problems in some cases. Randy had tried a fix in Draft 14, but it really was a stab in the dark. Tucker provided us with the following notes; also see 3.10.2(10/2) for additional details and background.

> Change mention of "name" to "function_call", as this is really the issue for parameters that are aggregates, allocators of an anonymous access type, etc. Name is used in too many contexts. In fact, we might be able to eliminate "expression and range" if we say "function_call (or equivalent operator invocation)".

Tucker suggests revising this paragraph as follows:

"... except in the case of a master: the execution of a body other than a package_body; the execution of a statement; or the evaluation of [an expression or range that is not part of an enclosing expression, name, range, or simple_statement] {a function_call (or equivalent use of an operator) that is not part of an enclosing function_call (or equivalent use of an operator) or assignment_statement."

Tucker claims that you do not want to wait for simple_statements other than assignments. It's quickly pointed out that procedure calls better be covered. Randy and Pascal wonder why we are changing this. Tucker says that you should clean up early in all other cases. That seems dubious. Eliminating range means you have to wait separately for each half; that seems bad.

Tucker is worried that parameters of function calls used in return statements would be finalized too late. Randy suggests just excepting simple_return_statement: "simple_statement other than simple_return_statement".

Steve Baird says that short circuits aren't included here, so we're wait twice. We don't want to say "short circuit". So we have to say expression. So we're nearly back to the original wording:

"... except in the case of a master: the execution of a body other than a package_body; the execution of a statement; or the evaluation of a expression, function_call, or range that is not part of an enclosing expression, function_call, range, or simple_statement other than a simple_return_statement."

Approve wording changes: 6-0-4.

7.6.1(11/2) AI-280/08

Erhard asks where the formal definition of when an object "ceases to exist" is. 13.11.2(10/2) defines it for Unchecked_Deallocation, but there appears to be nothing for "normal" objects. We also need to know this for types, because of the various rules for the Ada.Tags package.

Tucker suggests adding to the end of 7.6.1(11):

"After the finalization of a master is complete, the objects finalized as part of its finalization cease to *exist*, as do any types and subtypes defined and created within the master."

Index this under "exist" or possibly "exist; ceases to".

Approve addition: 8-1-1.

Steve Michell says that he wants “cease to exist” in italics. The rest of the group says that the term is “exist”. (Or do they just want to go to the brew pub? This was the last topic discussed before the meeting adjourned.)

8.3(12.3/2) AI-251/23

Corrected spelling of “arbitrarily”.

Approve correction: 11-0-0. [Draft13]

8.5.1(4.4/2) AI-423/08

The discussion of 12.4(8.4/2) (see that paragraph for details) decided that this paragraph needs to be reworded as:

“If the *object*_name denotes a generic formal object of a generic unit G, and the object_renaming_declaration occurs within the body of G or within the body of a generic unit declared within the declarative region of G, then the declaration of the formal object of G shall have a null_exclusion.”

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

8.5.4(4.2/2) AI-423/08

The discussion of 12.4(8.4/2) (see that paragraph for details) decided that this paragraph needs to be reworded similarly to 8.5.1(4.4/2).

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

9.3(2-3/2) AI-416/12

The changes to these paragraphs were unnecessary given the changes in AI-416 (declarations are not masters), so they were removed.

Approve correction: 9-0-1. [Draft14]

9.4(11.10-11.12/2) AI-397/08

Reorganized these to reflect the fact that there is no late “implemented-by” and added an AARM note to explain that.

Approve correction: 11-0-0. [Draft13]

9.5.2(13.1-13.3/2) AI-397/08

Reorganized these to reflect the fact that there is no late “implemented-by” and added an AARM note to explain that.

In 9.5.2(13.3/2), “then the operation” was changed to “then the entry” to match the preceding bullet.

Approve corrections: 11-0-0. [Draft13]

9.6.1(40/2) AI-351/10

Remove “Ada.” (see the discussion of G.3.1(32/2))

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

9.8(3/2) AI-345/11

The change here is now unnecessary because of changes in 3.9.4.

Approve correction: 11-0-0. [Draft13]

9.9(1/2) AI-345/11

The change here is now unnecessary because of changes in 3.9.4.

Approve correction: 11-0-0. [Draft13]

10.1.4(3/2) AI-217-6/17

Added “compilation” to the new wording to avoid confusion with “unit” standing alone.

Approve correction: 11-0-0. [Draft13]

10.2.1(9/2) AI-161/12

“which” should be “that”.

Approve correction: 11-0-0. [Draft13]

10.2.1(11.4/2) AI-161/13

Discussion of corrections in the following paragraphs caused Tucker to read this one. He immediately suggests removing “or protected” from 11.4. This rule doesn’t apply to derived protected types; the discriminants are the only thing that can change. But they have to be discrete or access, and thus always have preelaborable initialization. Also should change “type” to “extension”. “(in the case of a record extension)” because the discriminants don’t matter. “Moreover” should be “However”; the sense is an exception to the rule.

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion.although it was not originally part of these changes, but was added during the discussion. - ED]

10.2.1(11.5/2) AI-345/11

Added wording so that interfaces have preelaborable initialization. We surely want types derived from them to be able to have preelaborable initialization.

Approve correction: 11-0-0. [Draft13]

10.2.1(11.7/2) AI-161/12, AI-345/11

A package has basic_declarative_items, not declarative_items; and that was in the wrong font. Also, since protected interfaces are now protected types, this was changed to exclude protected interfaces.

Approve correction: 11-0-0. [Draft13]

10.2.1(15.4/2) AI-366/11

The previous version had multiple "not"s. The sentence was rewritten more like the wording in 4.8(5.3).

Approve correction: 11-0-0. [Draft13]

10.2.1(15.6) AI-366/11

Added "library" in front of "unit" to remove any ambiguity.

Erhard complains that "library unit" causes issues for nested declared pure packages. After some scrambling, we decide that you can't declare nested packages pure; only library units can be declared pure as the pragma can only be applied to library units. And an item in a nested package in a declared pure library package is still inside of that library package. Erhard withdraws his objection.

Approve correction: 11-0-0. [Draft13]

10.2.1(17/2) AI-366/11

The added sentence here missed a case: a limited private extension derived from a type with available stream attributes that has an access extension component. Such a type is streamable, and needs to be externally streamable to avoid problems with the Distribution Annex. This change introduces an unlikely incompatibility.

Approve correction: 11-0-0. [Draft13]

11.4.2(16/2, 18/2, 19/2, 21/2) AI-286/10

Remove "Ada." (see the discussion of G.3.1(32/2)).

Approve correction with changes: 11-0-0 [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

12.4(8.4/2) AI-423/08

The rewording of 8.5.1(4.4) should also have been carried out here.

This wording still confuses people. We try again to improve the wording. Ed suggests giving a name for the formal object as well. "..., where the actual denotes a generic formal object F of *G*, then the declaration of F shall have a null_exclusion;"

Randy asks whether the other similar paragraphs should be changed. No, they start with "if". But then this one is inconsistent. This paragraph should also start with "If": "If an instantiation occurs within the body…" Then "where" goes back to "and".

Erhard suggests: "If the actual denotes a generic formal object of another generic unit G, and the instantiation occurs within the body of G or within the body of a generic unit declared within the declarative region of G, then the declaration of the formal object of G shall have a null_exclusion."

Randy wonders aloud how this will work for renaming. We try rewording 8.5.1(4.4/2):

"If the *object*_name denotes the generic formal object of a generic unit G, and the object_renaming_declaration occurs within the body of G or within the body of a generic unit declared within the declarative region of G, then the declaration of the formal object of G shall have a null_exclusion."

This wording needs to be used in all four similar paragraphs: 8.5.1(4.4/2), 8.5.4(4.2/2), 12.4(8.4/2), and 12.6(8.2/2) (see those paragraphs).

Approve correction with changes: 11-0-0. [Draft13]

After the meeting it was noticed that references to "actual" and "instantiation" in 12.4(8.4) were somewhat ambiguous since we are talking about two different formals, so more words were added to qualify them:

"If the actual matching the formal_object_declaration denotes a generic formal object of another generic unit G, and the instantiation containing the actual occurs within the body of G or within the body of a generic unit declared within the declarative region of G, then the declaration of the formal object of G shall have a null_exclusion."

12.6(2/2) AI-260-2/07

The syntax needs to be written in the same order in this production as in the following productions.

Approve correction: 11-0-0. [Draft13]

12.6(8.2/2) AI-423/08

The discussion of 12.4(8.4/2) (see that paragraph for details) decided that this paragraph needs to be reworded similarly to 12.4(8.4/2).

Approve correction: 11-0-0. [Draft13]

12.6(18/2) AI-433/05

Changed the example to match AI-441.

Approve correction: 11-0-0. [Draft13]

13.1(13.1/2) AI-251/23

Added the rule discussed in e-mail for representation item conflicts.

Drop the latter two "progenitor or", as these are redundant. The implementation note should just say "one of the ancestors".

Approve correction with changes: 11-0-0. [Draft13]

13.2(6.1/2) AI-291-02/06

"which" was changed to "that".

Approve correction: 11-0-0. [Draft13]

13.12(7/2) AI-394/06

The "s" needs to be in the times font. (The syntax is "restriction"). No, this is talking about the concept, not the syntax. So, "restriction" should be in the normal font. This also applies to the AARM note.

Approve correction with changes: 9-0-1. [Draft14]

13.13.2(27.1/2) AI-251/23

Added a rule that the default version of 'Input is an abstract function for abstract types.

Approve correction: 11-0-0. [Draft13]

13.13.2(38/2) AI-251/23

Added a rule requiring any attribute specified for an interface type to be a null procedure.

Approve correction: 11-0-0. [Draft13]

13.13.2(51/2) AI-366/11

Reworded to avoid the non-technical use of "available", which is confusing here.

Steve Baird says that this paragraph should be bracketed. Tucker says that we should combine this with the following paragraph — this is the informal description, and the rest is the formal definition.

Approve correction with changes: 11-0-0. [Draft13]

A.10.7(17.3/2) AI-301/15

The first sentence was missing "the" and "this".

Approve correction: 11-0-0. [Draft13]

A.16(112/2) AI-248/15

Split the first sentence into two sentences.

Drop the commas from the first sentence. "Searches for entries matching Pattern in the directory named by Directory."

Approve correction with changes: 11-0-0. [Draft13]

A.16(129/2) AI-248/15

There is no such thing as "Is_Directory"; so this phrase was replaced by "Kind will never return Directory...".

Add "then" before Kind, and get rid of the second comma. Add "then" into the next note (130) as well.

Approve correction with changes: 11-0-0. [Draft13]

A.18.2(87/2) AI-302-3/15

Typo: "or" was changed to "are".

Approve correction: 11-0-0. [Draft13]

A.18.2(90/2) AI-302-3/17

Tucker proposed replacement wording for this paragraph; see the discussion in A.18.2(94/2).

Approve wording: 9-0-1.

A.18.2(94/2) AI-302-3/17

"not" was moved to improve the reading of this paragraph.

Tucker doesn't like this wording. He would like to drop most of it. Pascal and Randy disagree; we need some justification for these weird terms. Tucker is asked to come up with wording off-line.

Later in the meeting, Tucker shows his improved wording. He adds a paragraph in front of A.18.2(90/2), then shortens the headers of the bullets.

"Certain operations of this generic package have access-to-subprogram parameters. These operations disallow certain actions by the subprogram designated by such a parameter to ensure the operation is well-defined. In particular, certain operations disallow "tampering with cursors" of a container because they depend on the set of elements of the container remaining constant, and others disallow "tampering with elements" of a container because they depend on elements of the container not being replaced.

A subprogram is said to *tamper with cursors* of a vector object V if: ...

A subprogram is said to *tamper with elements* of a vector object V if: ..."

This has too many certains. Change some of them to "some":

"Redundant[Some operations of this generic package have access-to-subprogram parameters. These operations disallow certain actions by the subprogram designated by such a parameter to ensure the operation is well-defined. In particular, some operations disallow "tampering with cursors" of a container because they depend on the set of elements of the container remaining constant, and others disallow "tampering with elements" of a container because they depend on elements of the container not being replaced.]"

"Disallow" seems like a legality rule. "Prohibit" and "reject" are too strong.

Pascal suggests:

"Redundant[Some operations of this generic package have access-to-subprogram parameters. To ensure such operations are well-defined, these operations disallow certain actions by the designated subprogram. In particular, some operations disallow "tampering with cursors" of a container because they depend on the set of elements of the container remaining constant, and others disallow "tampering with elements" of a container because they depend on elements of the container not being replaced.]"

John suggests "guard against". Others suggest "check for". How about both?

"Redundant[Some operations of this generic package have access-to-subprogram parameters. To ensure such operations are well-defined, they guard against certain actions by the designated subprogram. In particular, some operations check for "tampering with cursors" of a container because they depend on the set of elements of the container remaining constant, and others check for "tampering with elements" of a container because they depend on elements of the container not being replaced.]"

Of course, all of the similar wording for other containers also should be changed.

Approve wording: 9-0-1.

A.18.2(136/2) AI-302-3/15

Missing boldfacing for "**all**".

Approve correction: 11-0-0. [Draft13]

A.18.2(151/2, 164/2, 177/2) AI-302-3/16

Dropped "or equal", since there's no reason to expand the vector if there is exactly enough space.

Approve correction: 9-0-1. [Draft14]

A.18.2(153/2) AI-302-3/15

Deleted a stray parenthesis.

Approve correction: 11-0-0. [Draft13]

A.18.2(163/2) AI-302-3/15

“Insert_Space” should have been “Insert” here.

Approve correction: 11-0-0. [Draft13]

A.18.2(191/2) AI-302-3/15

The wording here wasn’t quite the same as the wording in (193).

Approve correction: 11-0-0. [Draft13]

A.18.2(193/2) AI-302-3/15

Add a comma to “Otherwise, Swap” to be like 191.

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although there was no change to this paragraph in Draft 13 - ED]

A.18.2(207/2) AI-302-3/15

“otherwise, returns” should be “otherwise, it returns” to match other, nearby wording like A.18.2(201/2).

Approve correction: 11-0-0. [Draft13]

A.18.2(211/2) AI-302-3/16

“otherwise, returns” should be “otherwise, it returns” to match other, nearby wording like A.18.2(201/2).

Dropped parenthesis to be consistent with A.18.2(207/2).

Approve correction: 9-0-1. [Draft14]

A.18.2(215/2, 217/2, 219/2, 221/2) AI-302-3/15

“in the sense of” was replaced by “using”.

Approve correction: 11-0-0. [Draft13]

A.18.3(59/2) AI-302-3/15

Typo: “or” was changed to “are”.

Approve correction: 11-0-0. [Draft13]

A.18.3(61/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.3(66/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.3(107/2) AI-302-3/15

Add a comma to “Otherwise, Swap” to be like A.18.2(191).

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

A.18.3(109/2) AI-302-3/15

The wording wasn’t quite the same as the wording in A.18.3(107/2).

Approve correction: 11-0-0. [Draft13]

A.18.3(133/2, 135/2) AI-302-3/15

“in the sense of” was replaced by “using”.

Approve correction: 11-0-0. [Draft13]

A.18.4(7/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.4(12/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.4(17/2) AI-302-3/15

Typo: “or” was changed to “are”.

Approve correction: 11-0-0. [Draft13]

A.18.6(74/2) AI-302-3/15

A stray “the” was deleted.

Approve correction: 11-0-0. [Draft13]

A.18.7(3/2) AI-302-3/15

The paragraphs A.18.7(4/2) and A.18.7(5/2) in Draft 13 were just duplicate (and different) versions of this rule. They were deleted.

Approve correction: 11-0-0. [Draft13]

A.18.7(7/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.7(12/2) AI-302-3/17

Tucker improved this wording, see A.18.2(94/2) for the discussion.

Approve wording: 9-0-1.

A.18.7(16/2) AI-302-3/15

Typo: “or” was changed to “are”.

Approve correction: 11-0-0. [Draft13]

A.18.7(17/2) AI-302-3/15

“in the sense of” was changed to “using”.

Approve correction: 11-0-0. [Draft13]

A.18.7(22/2, 23/2) AI-302-3/15

Added To_Set, as requested in an Ada-Comment discussion.

John asks if there should there be a To_List? After all, there is a To_Vector. No, usually you are just adding an element to a list, whereas you might be using a set with one element in another operation (such as Union).

Approve correction: 11-0-0. [Draft13]

A.18.7(87/2) AI-302-3/15

“one” was changed to “a single”.

Approve correction: 11-0-0. [Draft13]

A.18.7(88/2, 89/2) AI-302-3/15

These were moved in front of A.18.7(90/2).

Approve correction: 11-0-0. [Draft13]

A.18.7(90/2) AI-302-3/15

A missing “to” was added.

Approve correction: 11-0-0. [Draft13]

A.18.8(8/2) AI-302-3/15

Added To_Set to this specification.

Approve correction: 11-0-0. [Draft13]

A.18.8(86/2) AI-302-3/15

“Equivalent_Element{s}”. “should” was changed to “is expected to” to match other wording.

Approve correction: 11-0-0. [Draft13]

A.18.9(9/2) AI-302-3/15

Added To_Set to the specification.

Approve correction: 11-0-0. [Draft13]

A.18.10(1/2) AI-302-3/15

Removed extra “does”.

Approve correction: 11-0-0. [Draft13]

A.18.11(1/2) AI-302-3/15

Removed extra “does”.

Approve correction: 11-0-0. [Draft13]

B.3.3(5) AI-216/17

“local_” is part of the syntax and shouldn’t be in italics.

Approve correction: 11-0-0. [Draft13]

C.7.1(18)

Erhard complains about the erroneous behavior of this paragraph. This has been expanded to the new packages. He thinks this is terrible for users, as they have no reliable way to determine if a task id represents a task that has ceased to exist. (See also the discussion of 7.6.1(11).)

Tucker says that something could be said in Annex H. He thinks this is appropriate as a high-integrity issue.

Ravenscar deals with this by avoiding task hierarchies, then the tasks never cease to exist.

It’s too late to do anything about this. The issue exists in Ada 95, but has not drawn any attention from the HRG or anyone else so far as we know. So it doesn’t seem to be a pressing concern.

C.7.3(5/2) AI-266-2/12

"**return**" should be in boldface.

Approve correction: 11-0-0. [Draft13]

D.1(20/2) AI-357/11

This paragraph was updated as suggested at the Burlington meeting.

Steve Michell is worried about this, because this no longer says that base priority is a source. But that's the reason for the change.

Joyce suggests:

"At any time, the active priority of a task is the maximum of all the priorities the task is inheriting at that instant. For a task that is not held (see D.11), its base priority is generally a source of priority inheritance, depending on the dispatching policy. Other sources of priority inheritance are specified under the following conditions:"

Where does the base priority get specified as inherited for the various policies? It doesn't seem to happen. Let's make this the default, and then have special wording to except EDF from it.

"At any time, the active priority of a task is the maximum of all the priorities the task is inheriting at that instant. For a task that is not held (see D.11), its base priority is a source of priority inheritance unless otherwise specified for a particular task dispatching policy. Other sources of priority inheritance are specified under the following conditions:"

Gary wonders about "under" the following conditions. "are present"? Nope.

We also need to update D.2.6(20/2) (see that paragraph).

Approve correction with changes: 11-0-0. [Draft13]

D.2.1(1.3/2) AI-355/09

Clarified as "task dispatching".

Approve correction: 11-0-0. [Draft13]

D.2.2(2.2/2) AI-355/09

The pragma was broken onto two lines so it fits.

This format looks ugly, but it has precedent (pragma Import, for instance). Indent this further though (it looks like one space).

Approve correction: 11-0-0. [Draft13]

D.2.2(3.4/2) AI-355/10

In the discussion on D.2.2(6.3/2) on Friday (see minutes of that paragraph), we became confused as to how the base and active priorities determine the policy that applies to a particular task. The problem is that whether or not the base priority is a source of priority inheritance depends on the policy.

Erhard says that he doesn't think it is problem. The one sentence (D.2.2(6.3/2)) talks about the base priority. All of the policies talk about what happens when the active priority is changed. In particular, we don't need to change D.2.2(6.3/2) to cover active priorities.

Pascal disagrees that there is no problem; while he agrees that D.2.2(6.3/2) is fine, he notes that there is nothing that says how you determine which policy applies to a particular task. We would hope that the active priority determines the policy, because we certainly don't want tasks with different policies on the same queue. But then, how is the active priority determined (given that it depends in part on the policy)?

Tucker suggests that the base priority determines how the active priority is calculated (that is, determines the policy that calculates the active priority); the active priority determines what queue you are on. Each queue is managed by a single policy. (This model was confirmed by Alan in e-mail on Sunday.)

Tucker notes that base priorities aren't changed until a protected action ends. So the exclusion guarantees still hold in a protected action — the active priority can't be changed by a change of base priority that changes the policy.

This paragraph should have separate sentences dealing with the base and active priorities:

"Tasks with base priorities within the range of priorities specified in a Priority_Specific_Dispatching pragma have their active priorities determined according to the specified dispatching policy. Tasks with active priorities within the range of priorities specified in a Priority_Specific_Dispatching pragma are dispatched according to the specified dispatching policy."

Approve wording change: 10-0-0.

Tucker worries that we need to change D.1(20/2). We did that previously (see discussion of D.1(20/2)).

The change was necessary so that base priority inheritance is the default for dispatching policies. We don't need any further change here.

D.2.2(6.3/2) AI-355/09

Changed to "immediately subject to the new dispatching policy"; added a note to explain that a bit.

Shouldn't this say something about "active priority"? If you inherit outside out of EDF, the task is no longer dispatching with EDF, right? Right.

"A task that has its active priority changed may move from one dispatching policy to another. It is immediately subject to the new dispatching policy."

Steve Michell thinks that both words are needed. We're confused. But Steve Baird and Randy wonder about EDF, where the base priority can change but the active priority doesn't depend on it.

We need to get Alan's input; we'll defer the topic until then.

(Further discussion on this topic is recorded under paragraph D.2.2(3.4/2). Ultimately, we left this wording as originally corrected.)

Approve correction: 11-0-0. [Draft13]

D.2.2(13.1/2) AI-333/07

Rewrote as assigned in Burlington.

Approve correction: 11-0-0. [Draft13]

D.2.3(10/2) AI-333/07

Rewrote as assigned in Burlington.

Approve correction: 11-0-0. [Draft13]

D.2.4(8/2) AI-298/06

Jean-Pierre would like a user note added to explain why this is non-preemptive.

"Unlike policy FIFO_Within_Priorities, these events are not task dispatching points."

D.2.4(8/2) is the only non-blocking event that is a task dispatching point. Tucker suggests changing its last sentence:

"This is the only non-blocking event that is a task dispatching point." This needs to be clear that it is only talking about this policy. Perhaps "for this policy" should be added.

Better would be to drop this sentence from D.2.4(8), and add after the bullets:

"A non-blocking delay_statement is the only non-blocking event that is a task dispatching point (see D.2.1)."

Approve change: 8-0-2.

After the meeting it was noted that the added sentence could be construed as applying in general, so it was changed to:

"For this policy, a non-blocking delay_statement is the only non-blocking event that is a task dispatching point (see D.2.1)."

D.2.4(9/2) AI-333/07

Rewrote as assigned in Burlington.

Approve correction: 11-0-0. [Draft13]

D.2.4(10/2) AI-298/06

Jean-Pierre wonders about "contain" in this paragraph, as pragmas "apply". Tucker suggests "…provided the associated protected unit…".

Approve change: 8-0-2.

D.2.5(6/2) AI-355/09

"single {priority} level".

Add "priority" in front of "levels", too.

Approve correction with change: 11-0-0. [Draft13]

D.2.5(9/2) AI-355/09

The wording was corrected to be able to handle a range of priorities.

Remove the comma.

Approve correction with change: 11-0-0. [Draft13]

D.2.5(14/2) AI-355/09

The rule D.2.5(14/2) in Draft 13 was worded oddly; once it was reworded, it was clear that this was already covered by the FIFO. So the rule was completely redundant, and it was removed and replaced by an AARM note.

Approve correction: 11-0-0. [Draft13]

D.2.5(15/2) AI-333/07

Rewrote by John and Alan as assigned in Burlington.

Approve correction: 11-0-0. [Draft13]

D.2.6(1/2) AI-357/12

This introduction was garbage, because it was not clear that it applies only to certain dispatching policies. (The deadline has no effect on the dispatching of most policies.) Randy had reworded it as:

"The deadline of a task is an indication of the urgency of the task; it represents a point on an ideal physical time line. For policies that use deadlines, whenever tasks compete for processors or other implementation-defined resources, the resources are allocated to the task with the earliest deadline. "

"…and might affect how resources are allocated to the task."

Steve Baird complains that the paragraph is still too specific; what if there is a Latest_Deadline_First policy?? Just drop the second sentence. We are left with: "The deadline of a task is an indication of the urgency of the task; it represents a point on an ideal physical time line and might affect how resources are allocated to the task".

Approve correction with changes: 9-0-1. [Draft14]

After the meeting, this wording was revised slightly for clarity:

"The deadline of a task is an indication of the urgency of the task; it represents a point on an ideal physical time line. The deadline might affect how resources are allocated to the task."

D.2.6(6.1/2) AI-357/13

Drop **in** from the specification of Get_Deadline.

We give the editor freedom to remove **in** from any function specification.

This paragraph shouldn't have an inserted paragraph number; this is a new clause. That's also true for D.2.6(2.1/2-2.2/2).

Approve change and permission: 10-0-0.

D.2.6(13/2) AI-357/11

This was changed to use "protected action", as this requires the same rule as D.5.1(10).

Get rid of "a task's deadline" by changing to "the deadline of a task". (Do this also in D.2.6(2/2) and D.2.6(29/2).)

Approve correction with change: 11-0-0. [Draft13]

D.2.6(15/2-16/2) AI-357/13

Steve Michell says that that the bullets D.2.6(15-18) should be describing events; these things should only happen when an task is added to the ready queue.

These bullets are talking about when you lose inheritance from a protected object. They're not intended to describe adding another task to the ready queue.

Tucker says that we could add “any time” to this to make it more event-like:

“any time there is a task…”

“any time there is a non-empty…”

Erhard disagrees with this change. He explains that this is the invariant for EDF; a task dispatching point occurs at any point when that invariant would be violated. So it’s not really an event.

Joyce argues that for EDF every clock tick is really an event, so it is perfectly legitimate to consider the phrase “any time” as talking about events.

Pascal says that he likes this viewpoint; he also notes that we’ve reworded this several previous times.

The problem is that “whenever” seems to imply “at any time”. But that doesn't apply to the first bullet (because it only happens when an explicit call is made), nor does it apply to the other bullets (because they can only happen when the active priority of a task is changed [usually by entering or exiting a protected action]).

Remove “whenever” from paragraph 15. Add “when” to paragraph 16.

Approve changes: 9-0-1.

D.2.6(20/2) AI-357/13

We need to add wording that says that the base priority is not a source of priority inheritance (see the discussion of D.1(20/2).

“For a task *T* to which policy EDF_Across_Priorities applies, the base priority is not a source of priority inheritance; the active priority when first activated or while it is blocked is defined as the maximum of the following:”

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although there was no change to this paragraph in Draft 13 - ED]

D.2.6(25/2) AI-357/12

Add “ready” so we’re not putting blocked tasks on the ready queue.

Approve correction: 9-0-1. [Draft14]

D.2.6(26/2) AI-357/11

“this package” was changed to “Ada.Dispatching.EDF”

Drop “Ada.”, we don’t put that in front of package names in the Standard.

Approve correction with change: 11-0-0. [Draft13]

D.3(6/2) AI-327/10

The models of D.3 and D.5.2 were different; the wording was changed to reconcile that. Now, protected objects always have a priority, regardless of the locking policy.

Drop “While”. Make the part that started with “while”, up to the first comma, a sentence. “These rules” should be “The locking policy specifies…”

Approve correction with changes: 11-0-0. [Draft13]

D.3(6.1/2) AI-327/10

Hoisted this rule so that all protected objects can use these pragmas, no matter what locking policy is used.

Approve correction: 11-0-0. [Draft13]

D.3(8/2-11/2) AI-327/10

Defined the initial (not ceiling) priority for protected objects, and made it the same as the one set in D.5.2.

Approve correction: 11-0-0. [Draft13]

D.5.1(10/2) AI-188-2/03

The wording left it unclear which task needed to be outside of a protected region. The wording also mysteriously changed from “protected action” to “abort-deferred region”. This change was unintentional and not needed, and it was removed.

“that T” should be “when T”.

Approve correction with change: 11-0-0. [Draft13]

D.5.1(12.1/2) AI-188-2/04

Pascal complains about the possessive here; change to “the priority of a task”.

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

D.5.1(15/2) AI-321/06

Used the name of the policy, as there is no “standard” policy anymore.

Approve correction: 11-0-0. [Draft13]

D.5.2(4/2) AI-327/10

Reworded to match the revised model of D.3, so that the priority always has a defined value.

Bracket the last part (the comma and after), as it follows from the fact that the attribute represents a variable. Humm, paragraph 3 needs to mention that it is a variable. “non-aliased variable component”. No, that doesn’t work; if the prefix is constant, the attribute has to be constant as well. So add to paragraph 3 as the penultimate sentence: “P'Priority denotes a variable if and only if P denotes a variable.”

Approve correction with changes: 11-0-0. [Draft13]

D.8(26/2) AI-432/02

There were two extra “and”s in the text.

Approve correction: 11-0-0. [Draft13]

D.9(14/2) AI-355/09

Deleted the note as discussed at the Burlington ARG meeting.

Approve correction: 11-0-0. [Draft13]

D.11(4-6) AI-357/13

The semantics of a held task don't work for EDF, because the base priority doesn't have any role in determining the active priority. Reword these rules to be more explicit.

For D.11(5):

"The Hold operation sets the state of T to held. For a held task, the active priority is reevaluated as if the base priority of the task were the held priority."

For D.11(6):

"The Continue operation resets the state of T to not-held; its active priority is then reevaluated as determined by the task dispatching policy associated with its base priority."

Should we specify the task dispatching policy for the held priority? Yes. If the EDF policy applies to the entire system, the base priority doesn't have any effect on the active priority, and this model might break.

Add a new paragraph after D.11(4): "For any priority below System.Any_Priority'First, the task dispatching policy is FIFO_Within_Priorities.

AARM to be honest: This applies even if a Task_Dispatching_Policy specifies the policy for all of the priorities of the partition.

AARM ramification: A task at the held priority never runs, so it is not necessary to implement FIFO_Within_Priorities for systems that have only one policy (such as EDF_Across_Priorities).

Capitalize "Integer" in D.11(4).

Approve changes: 10-0-0.

D.13(2/2) AI-249/12

The closing "}" was missing.

Approve correction: 11-0-0. [Draft13]

D.13.1(2/2) AI-249/12

"names" was changed to "is".

Approve correction: 11-0-0. [Draft13]

D.14(13/2) AI-307/13

The CPU_Time is set to zero when the task is created. The normative wording that explains the non-determinism was changed to an AARM note.

Approve correction: 11-0-0. [Draft13]

D.14.1(4/2) AI-307/13

The discriminant is now a **not null access constant**.

Approve correction: 11-0-0. [Draft13]

D.14.1(7/2) AI-307/13

The mode of Cancelled should be **out**.

Approve correction: 11-0-0. [Draft13]

D.14.1(11/2) AI-307/13

Added missing hyphen in “execution time overrun”.

Approve correction: 11-0-0. [Draft13]

D.14.1(15/2) AI-307/13

The rewritten version proposed at the Burlington ARG meeting was used.

Approve correction: 11-0-0. [Draft13]

D.14.1(21/2) AI-307/13

Rewrote this paragraph to make it clearer what priority is involved.

Approve correction: 11-0-0. [Draft13]

D.14.1(22/2) AI-307/13

This paragraph is new; it gives a previously missing rule.

Approve correction: 11-0-0. [Draft13]

D.14.1(23/2 and 25/2) AI-307/13

“TM.T” was changed to “TM.T.**all**” in both paragraphs.

Approve correction: 11-0-0. [Draft13]

D.14.2(21/2) AI-354/10

Rewrote this paragraph as discussed at the Burlington ARG meeting.

“; the tasks continue to execute.” should be “. Nevertheless, the tasks continue to execute.” “…counting down{, unless exhausted}”.

Approve correction with changes: 11-0-0. [Draft13]

D.14.2(29/2) AI-354/10

Rewrote this paragraph to make it clearer what priority is involved.

Approve correction: 11-0-0. [Draft13]

D.15(3/2-7/2) AI-297/14

Made parameters line up. Broke up into several paragraphs as requested at the Burlington ARG meeting.

The Cancelled parameter of Cancel_Handler should be lined up.

Approve correction with change: 11-0-0. [Draft13]

D.15(26/2) AI-297/15

Erhard notes that this should say “potentially blocking operation”.

Approve change: 10-0-0.

E.2.2(8/2) AI-366/11

The wording was too broad; it included limited types without stream attributes, which the original wording did not. Added “any available” to eliminate the problem.

Approve correction: 11-0-0. [Draft13]

G(5/2-6.1/2) AI-296/11

This list needed to include the vector and matrix packages.

Approve correction: 11-0-0. [Draft13]

G.1.2(20/2) AI-185/05

Remove “Ada.” (see the discussion of G.3.1(32/2)).

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

G.2.2(3.2/2) AI-256/10

Replaced “it” by T'Model_Mantissa.

Approve correction: 11-0-0. [Draft13]

G.3.1(24/2) AI-296/11

Adding missing space between X and :.

Approve correction: 11-0-0. [Draft13]

G.3.1(32/2) AI-296/12

Remove “Ada.”. The editor should do a global search for “Ada.” and remove any on package names in running text.

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

G.3.1(33/2) AI-296/11

Changed to “subprograms”, because there is at least one procedure in this package.

Approve correction: 11-0-0. [Draft13]

G.3.1(34/2) AI-296/11

“involve an inner product” needed to be in italics.

Approve correction: 11-0-0. [Draft13]

G.3.1(58/2, 64/2, 66/2) AI-296/11

Dropped the extra word “matrix”.

Approve correction: 11-0-0. [Draft13]

G.3.1(67/2) AI-296/11

Added missing space between X and :.

Approve correction: 11-0-0. [Draft13]

G.3.1(68/2, 70/2, 72/2) AI-296/11

Changed the result range as demanded by Pascal.

Approve correction: 11-0-0. [Draft13]

G.3.1(90/2) AI-296/11

“may” was changed to “should”.

Approve correction: 11-0-0. [Draft13]

G.3.2(46/2) AI-296/11

Added missing space between X and :.

Approve correction: 11-0-0. [Draft13]

G.3.2(54/2) AI-296/12

Remove “Ada.” (see the discussion of G.3.1(32/2)).

Approve correction with changes: 11-0-0. [This was voted on with the Draft13 changes, although it was not originally part of these changes, but was added during the discussion. - ED]

G.3.2(56/2) AI-296/11

"involve an inner product" needed to be in italics.

Approve correction: 11-0-0. [Draft13]

G.3.2(58/2, 60/2, 62/2, 66/2, 92/2, 94/2, 96/2, 100/2) AI-296/11

"Cartesian" should be capitalized.

Approve correction: 11-0-0. [Draft13]

G.3.2(90/2) AI-296/11

Added spaces.

Approve correction: 11-0-0. [Draft13]

G.3.2(112/2, 122/2) AI-296/11

Dropped the extra word "matrix".

Approve correction: 11-0-0. [Draft13]

G.3.2(135/2) AI-296/11

Added missing space between X and :.

Approve correction: 11-0-0. [Draft13]

G.3.2(136/2, 138/2, 140/2) AI-296/11

Changed the result range as demanded by Pascal.

Approve correction: 11-0-0. [Draft13]

G.3.2(148/2) AI-296/11

Added spaces.

Approve correction: 11-0-0 [Draft13]

G.3.2(151/2) AI-296/11

Boldfaced **abs**.

Approve correction: 11-0-0. [Draft13]

G.3.2(160/2) AI-296/11

"may" was changed to "should". Boldfaced **abs**.

Approve correction: 11-0-0. [Draft13]

H.4(2/2) AI-394/04

The operative part of this paragraph was moved to Implementation Requirements and the rest deleted.

Approve correction: 11-0-0. [Draft13]

H.4(3/2) AI-394/04

This was changed to the standard introduction for new restrictions.

Approve correction: 11-0-0. [Draft13]

H.4(23.1/2-23.6/2) AI-394/04

The restrictions that need to be supported here are listed as a bulleted list.

Drop the normative text for H.4(23.3), make it a "user note".

NOTES
10 Uses of *restriction_parameter*_identifier No_Dependence defined in 13.12.1: No_Dependence => Ada.Unchecked_Deallocation and No_Dependence => Ada.Unchecked_Conversion may be appropriate for high-integrity systems. Other uses of No_Dependence can also be appropriate for high-integrity systems.

Copy the AARM notes after the user note.

Make 23.5 into a list of sub bullets:

- the following uses of *restriction_parameter*_identifiers defined in D.7[, which are checked prior to program execution]:
    - Max_Task_Entries => 0,
    - Max_Asynchronous_Select_Nesting => 0, and
    - Max_Tasks => 0.

Approve correction with changes: 11-0-0. [Draft13]

After the meeting it was noted that the above bullet is not the last element in its list of bullet. So the last bullet was moved in front of this one, and "; and" added to it:

- the **pragma** Profile(Ravenscar); and

H.6(5/2) AI-265/09

This paragraph was rewritten with additional fixes, including ensuring that there is one policy per partition.

Approve correction: 11-0-0. [Draft13]

J.9(3/2) AI-345/11

We don't want to allow specifying Storage_Size on a task interface or a class-wide task type; this wording change is needed because interfaces are now task types.

Approve correction: 11-0-0. [Draft13]

J.13 AI-394/05

Change the title of this clause to drop the "s" from Restrictions. (These are "*restriction*_identifiers")

Approve correction: 9-0-1. [Draft14]

J.13(1/2) AI-394/06

The "s" needs to be in the times font. (The syntax is "restriction".) No, this is talking about the concept, not the syntax. So, "restriction" should be in the normal font.

Approve correction with changes: 9-0-1. [Draft14]

J.13(2/2) AI-394/05

Change the lead-in to drop the "s" from Restrictions.

Approve correction: 9-0-1. [Draft14]

J.13(3/2-5/2) AI-394/05

These three items should be consistent with each other, thus they were reworded to:

No_Asynchronous_Control
  Semantic dependence on the predefined package Asynchronous_Task_Control is not allowed.

No_Unchecked_Conversion
  Semantic dependence on the predefined generic function Unchecked_Conversion is not allowed.

No_Unchecked_Deallocation
  Semantic dependence on the predefined generic procedure Unchecked_Deallocation is not allowed.

Approve correction: 9-0-1. [Draft14]