Minutes of the 32nd ARG Meeting

1-3 June 2007

Paris, France

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (midday Friday only), Bibb Latting, Pascal Leroy, Brad Moore, Erhard Ploedereder , Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Bill Thomas (except Friday morning and Sunday).

Observers: Greg Gicca (except Sunday), Sergey Rybin (except Sunday).

Meeting Summary

The meeting convened on 1 June 2007 at 09:15 hours and adjourned at 15:40 hours on 3 June 2007. The meeting was held in two different conference rooms at the offices of AdaCore in Paris, France. The meeting covered most of the agenda (12 Ada 2005 AIs were not covered: 22, 23, 26, 32, 33, 36, 38, 39, 41, 44, 47, and 48).

AI Summary

The following AIs were approved:

AI05-0008-1/04 General access values that might designate constrained objects (6-0-4)
AI05-0037-1/01 Out of range <> associations in array aggregates (10-0-0)

The following AIs were approved with editorial changes:

AI05-0002-1/02 Unconstrained arrays and C interfacing (11-0-0)
AI05-0017-1/02 Freezing and incomplete types (7-0-3)
AI05-0019-1/02 Primitive subprograms are frozen with a tagged type (7-0-3)
AI05-0024-1/03 Run-time accessibility checks (4-0-6)
AI05-0028-1/04 Problems with preelaboration (10-0-0)
AI05-0035-1/02 Inconsistencies with pure units (8-0-2)
AI05-0040-1/01 Limited with clauses on descendants (10-0-0)
AI05-0043-1/00 The Exception_Message for failed language-defined checks (8-0-2)
AI05-0046-1/01 Null exclusions must match for profiles to be fully conformant (10-0-1)
AI05-0055-1/01 Glitch in EDF protocol (10-0-2)
AI05-0056-1/01 Wrong result for Index function (8-0-2)

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

AI05-0009-1/03 Confirming rep. clauses and independence (7-0-3)
AI05-0030-1/01 Requeue on synchronized interfaces (10-0-2)
AI05-0034-1/01 Categorization of limited views (9-0-1)
AI05-0045-1/01 Termination of unactivated tasks (12-0-0)
AI05-0050-1/01 Return permissions are not enough for build-in-place (10-0-1)
AI05-0053-1/01 Aliased views of unaliased objects (10-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0042-1/00 Overriding versus implemented-by
AI05-0049-1/00 Extend file name processing in Ada.Directories
AI05-0051-1/01 Accessibility of dispatching function calls (aka another Baird question)
AI05-0052-1/01 Coextensions and distributed overhead
AI05-0054-1/01 Variable views of constant objects
AI05-0057-1/00 The class attribute of a constrained subtype

The following AI was tabled for lack of agreement on how to proceed:

AI05-0012-1/02 Independence and Representation Clauses for atomic objects

The following AI was voted No Action:

AI05-0018-1/02 Formal package matching rules (8-1-1)

The following SIs were approved:

SI99-0003-1/03 Support overriding indicators (11-0-0)

The following SIs were approved with editorial changes:

SI99-0009-1/03 Support new aggregate features (11-0-0)
SI99-0026-1/01 Baseline version of ASIS standard (10-1-0)
SI99-0027-1/01 Obsolescent features should have their own annex (8-0-2)
SI99-0028-1/01 What does appropriate kinds mean? (9-0-2)

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

SI99-0004-1/03 Changes to Asis for changes to access types (10-0-1)
SI99-0012-1/04 Add support for null procedure (8-0-2)
SI99-0022-1/02 Add Boolean queries to ease use of trait kinds (9-0-1)
SI99-0025-1/01 ASIS 99 allows too much variability between implementations (10-0-1)

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

SI99-0023-1/01 Usages of subtypes Name and Name_List in the ASIS specifications
SI99-0024-1/05 Provide a semantic model in additional to existing syntactic model
SI99-0029-1/01 Inconsistent inconsistent list

The following SI was voted No Action:

SI99-0016-1/03 Correct Corresponding_Body (10-0-1)

Detailed Minutes

Meeting Minutes

John describes a number of errors he found in the minutes of the previous meeting:

In the discussion of AI05-0009-1, change “IA” to “Implementation Advice”.

Also in the discussion of AI05-0009-1, the paragraph starting with “Should this go in Annex C...” needs a period at the end.

In the fourth paragraph of the discussion of AI05-0017-1, add a missing ‘s’: “Taft Amendment type{s}”.

In the discussion of AI05-0020-1, the paragraph starting “Subject should be...” needs a period at the end.

In the first paragraph of the discussion of AI05-0023-1, drop an extra ‘t’: “No[t]”.

In the paragraph starting “We check Gary’s example...” in the discussion of AI05-0024-1, drop one of the “the”s: “the revised [the] rule”, and change “in given” to “as given”.

The minutes of the previous meeting were then approved with changes by acclamation.

Date and Venue of the Next Meeting

Next meeting will be after the SIGAda conference: Washington DC, Nov 9-11, 2007.

We tentatively add (see discussion under Schedule of ASIS) an extra meeting in Tampa, Feb 8-10, 2008, with Greg Gicca hosting.

ACATS test objective review plan

Randy reminds the ARG that they have oversight of the ACATS development. The ARA is currently having him create a new ACATS version for Ada 2005. The plan is to have a first official new ACATS release for Ada 2005 in the fall. He is providing snapshots of the new tests; the current snapshot is the second one. There will be at least one more snapshot before the release is generated.

The ARG will need to review in some fashion the test objectives and tests covered. Randy is developing test objectives for the entire language, concentrating on subclauses with the most changes in the Amendment.

Randy notes that so far he has created objectives for 26 subclauses with 408 paragraphs, which has made 540 objectives. Of those, 275 are at least partially covered by tests (62 objectives are covered by new tests in the latest snapshot); 344 need additional testing. Many of these are fairly low priority (untested functionality which is commonly used and unlikely to be wrong; additional cases for existing tests — like protected subunits; and objectives for which testing is impractical).

A complete formal review of the developed test objectives is out of the question, the volume is just too large. So how should we do a review? After discussion, we decide on a two-pronged plan:

- Randy should circulate the current objective documents, in whatever format is convenient. (They're in Excel spreadsheets at the moment.)
- Randy will prepare a list of objectives covered by new tests for a vote. We'll have a letter ballot on the whole (with members being able to ask for separate consideration of specific objectives).

Randy should try to initiate the ballot in September or so, in order that the ballot can be completed before the full release is made.

ASIS standard format (adherence to the drafting document)

Randy reads all of his research.

Font usage in the current standard is OK. ITTF requires "standard" fonts, but curiously does not list Courier. We're not going to change the fonts we use (Courier surely seems "standard").

We're not supposed to have single subclauses (if there is a 6.5.1, there ought to be a 6.5.2). We're also not supposed to have text in a (sub)clause that has subclauses (these are called hanging paragraphs). Hanging paragraphs and single subclauses should be fixed. Tucker suggests of starting numbering with 0; Randy notes that the drafting rules do not appear to disallow this. This reordering/numbering will be done last in the process.

The names of examples and notes are not properly noted in the original ASIS standard. The formatting tool has been updated to the handle this properly. But not all examples and notes are marked as such in the original text. We'll fix those as we find them. Similarly, the formatting tool has been updated to properly handle numbered/lettered lists.

The drafting standard says that a definitions section is optional, but it doesn't make clear whether that is because having any definitions is optional or whether splitting them out is optional. (The Ada standard assumes the latter.) We agree that making a separate definitions section is important, ASIS doesn't have many definitions anyway. Which clause is it? Some standards have it in 2 and some in 4. Anyway, it should be a clause near the top. Actually, it could replace the current 1.3. Greg will decide (and create this section and an SI to match).

Randy needs to fix the numbering of tables and figures (in the formatting tool).

The wording rules will be checked on a case-by-case basis. We already follow most of them anyway.

Schedule of ASIS

Pascal points out that we had discussed in Porto that we needed to have most of the technical work done this summer. That doesn't seem to be possible.

How do we get the work moving? Tucker suggests more/earlier deadlines; he claims to be deadline driven. Pascal says that he tried to get Tucker to get the work done earlier this time, and it didn't work. Tucker says that he needs **real** deadlines.

Maybe we need an extra meeting? That would provide an extra hard deadline for Tucker. Pascal says that should be probably in the winter. We discuss a site; Steve offers Cupertino, Greg suggests Tampa. Tampa seems like a good idea. We select the dates of Feb 8-10, 2008.

Thanks

By acclamation, the ARG thanks our host, AdaCore, for the accommodations for the meeting and for the sandwiches on Sunday.

Old Action Items

Pascal did not do his SIs or SI research (he was waiting for SI99-0024 to be completed). Tucker did not do AI05-0027 and the lower priority items AI05-0003 and AI05-0006. All other items were completed.

New Action Items

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

Steve Baird:

- AI05-0050-1
- AI05-0051-1 (with Tucker Taft)
- AI05-0053-1
- AI05-0057-1
- AI split from AI05-0051-1 to handle second question (see discussion of AI05-0051-1) (with Tucker Taft)
- Create an SI outlining problems and possible solutions in the ASIS.Data_Decomposition package (see discussion of Jean-Pierre's request to find Size).

Randy Brukardt:

- AI05-0034-1
- AI05-0045-1
- AI05-0049-1
- SI99-0004-1
- SI99-0012-1
- SI99-0025-1
- Update the formatting tool to handle numbering of tables and figures, and to create a table of contents for them (see ASIS standard format).
- Circulate ACATS test objectives to interested parties.
- Create a listing of test objectives covered by new ACATS tests for a letter ballot by the ARG starting in September or October.

Editorial changes only:

- AI05-0002-1
- AI05-0017-1

- AI05-0018-1
- AI05-0019-1
- AI05-0024-1
- AI05-0028-1
- AI05-0035-1
- AI05-0040-1
- AI05-0043-1
- AI05-0046-1
- AI05-0055-1
- AI05-0056-1
- SI99-0009-1
- SI99-0026-1
- SI99-0027-1
- SI99-0028-1

Alan Burns:

- AI05-0030-1

Greg Gicca:

- Add newly ARG approved SIs to the draft ASIS standard.
- Create a definitions section for the draft ASIS standard and a matching SI (see ASIS standard format).
- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format)

Bibb Latting:

- AI05-0009-1

Pascal Leroy:

- AI05-0013-1 (see private editorial review e-mail)
- AI05-0042-1
- AI05-0052-1
- AI05-0054-1
- SI99-0007-1 (move to the semantic domain)
- SI99-0019-1
- SI99-0021-1 (move to the semantic domain)
- SI99-0023-1
- SI99-0029-1
- Study whether it is possible to properly define "=" for types in ASIS rather than declaring "=" abstract - create an SI to fix this if possible (see Details of the ASIS Revision in the Albuquerque minutes)

Tucker Taft:

- AI05-0003-1 (lower priority)
- AI05-0006-1 (lower priority)
- AI05-0027-1

- AI05-0051-1 (with Steve Baird)
- AI05-0054-2 (version to make this non-erroneous)
- AI split from AI05-0051-1 to handle second question (see discussion of AI05-0051-1) (with Steve Baird)
- SI99-0024-1

Bill Thomas:

- SI99-0022-1

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment AIs. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have six votes for, one vote against, and two abstentions.

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

Detailed Review of ASIS Issues

SI99-0003-1/03 Support overriding indictors

Sergey does not like this design, because kinds are used mainly with elements. He would prefer Boolean query functions.

The issue is that this is a three state problem; Boolean does not map well to that.

After further discussion, Sergey drops his objection, because parameter modes are handled this same way. (They’re another case with multiple states.)

Randy notes that we approved this last time, but it was sent back because an item of Not_An_Overriding_Indicator is needed to be consistent with other ASIS “kinds”.

Approve SI: 11-0-0.

SI99-0004-1/03 Changes to Asis for changes to access types

Tucker tries to explain how the SI works now.

Pascal suggests that the result of Anonymous_Access_To_Object_Subtype_Mark be ASIS.Name. Returns are not controversial and should always be done.

There is a trait here, didn’t we decide to drop all changes to those? Let’s discuss that later as part of SI99-0022-1 (note the resolution given in that SI).

Sergey suggests that Object_Declaration_View could have the semantics of Object_Declaration_Subtype; he wants existing programs to work even with Ada 2005. That’s not possible in general; programs will get unexpected returns and fail. He argues that cases that are incompatible aren’t possible in Ada 95, so there is no problem with this function.

Erhard asks if we need to be “bug compatible”? View is an Ada technical term, it is abused here. Pascal suggests a rename to provide the old name (and share the semantics). However, he then goes on to worry that there might exist applications that depend on getting Not_A_Definition, where they won’t get those anymore.

Tucker thinks that changing the values that are returned are much more likely to compromise compatibility than just changing names of functions. If a program gets an unexpected return, it will fail in strange ways.

Pascal notes that “View” will be used in its Ada sense in the semantic interface; that will be very confusing.

Why does this function return Nil_Element for tasks? That was just copied from the old ASIS. It would be better if this was corrected to return something that would be marked as implicit. Someone looks up the function for determining that; it is Is_Part_of_Implicit.

Sergey says that implicit items are never created for purely syntactic queries. It is important that this is not changed. We wonder about similar cases. ASIS doesn’t provide a way to find out **private** for an empty private part or **declare** if there are no declarations. So why is `task T;` different from `task T is end;`. But this isn’t true: There are queries for **private** and **declare** (see 16.38 and 18.14).

Tucker proposes that this returns a task definition, and we add a query that to tell if the **end**; is present. It should be named Is_Task_Definition_Explicit. As a general principle, Nil_Element should not convey semantic information.

Straw poll on this idea: 10-1-3. (Sergey opposes, doesn’t like the extra query as it changes existing ASIS behavior.)

So remove “Returns a Nil_Element for a single_task_declaration that has no explicit task_definition.”, add Is_Task_Definition_Present (which returns True only for an definition element). This name is more consistent. That leads to much discussion.

The whole SI needs to be written using the new format and new standard (this mainly means removing the comment symbols before every line).

For the function Result_Subtype, the old description was left accidentally. Delete the old description; that is the entire paragraph that starts “Returns the subtype_mark expression...”

Approve intent of SI: 10-0-2.

Randy will update this SI.

SI99-0009-1/03 Support new aggregate features

Sergey argues that his original idea is better. His argument is that <> is not an expression. He complains that you have to construct an enclosing element. Pascal says that a good thing; once you get a Nil_Element, you can’t go anywhere. You can’t continue to an outer element or a next element.

Tucker points out that with Sergey’s idea, code will break whenever it gets a <>. And it will break at runtime, and not in an obvious way. There are many contexts where a <> is not allowed, and that also true for other kinds of expressions. Make sure that Box_Expression is last. Tucker suggests that maybe this should be second. Second is somewhat nicer, because then the range of normal expressions is contiguous.

Sergey asks what is the type of <>. It has the component type when the aggregate is normalized, does not have a type when it is not normalized (it returns Nil or raises an exception). That should be said somewhere. For array aggregates, you can always get the type of <> (it is the type of the component of the array).

Add a bullet to the list of 17.1: “A_Box_Expression returned by Component_Expression applied to an unnormalized record association.” [Editor’s note: this change will need to be reflected in the semantic interface, as 17.1 is the function Corresponding_Expression_Type, which will probably be obsolesced by the semantic interface.]

Sergey would rather that a <> never has a type, as that would be easier to implement. We’re not terribly interested in what is easier to implement, but rather what is more useful for users (assuming it is well-defined).

John wants the second last bullet to end with a period rather than a semicolon.

Correct typos: “betore” should be “before”. “a [a] box expression”.

Approve AI with changes: 11-0-0.

SI99-0012-1/04 Add support for null procedure

Has_Null should be named Is_Null_Procedure

We're doing this the same way as abstract subprograms.

Jean-Pierre wonders whether this routine should raise an exception. No, predicates always return False and never raise an exception.

With this name change, we seem to need Is_Abstract_Subprogram (even though you could say Has_Abstract), because you could have types that have **abstract**.

Approve intent of this SI: 8-0-2.

Randy will update this SI.

SI99-0016-1/03 Correct Corresponding_Body

This is now handled by SI99-0025-1.

No action: 10-0-1.

SI99-0022-1/03 Add Boolean queries to ease use of trait kinds

Typos: Capitalize Boolean in the subject of the SI. The first line of the summary has 'an' instead of 'a'. Function names and type names should be capitalized. There is an unclosed parenthesis in the summary. Get rid of extra spaces.

Change Has_Limited description to:

"Returns True if the Element includes the reserved word **limited**."

The other functions need a similar description.

Should there be a Has_Tagged function? We don't have to have one, as that can be determined other ways. But it seems more consistent to have it; Tucker says that having it could be simplifying.

Should Has_Private work on private packages? (Yes.) What about private parts? (No, because that is a not a separate syntactical option.)

Jean-Pierre notes that since these always return False for unusual elements, the parameter types of all Boolean queries take an element (rather than one of the subtypes of element).

Tucker complains that "includes" is too nebulous, it could be thought to include things in inner items. Change the wording to use "appears", and possibly give a reference to the syntactic categories of the Ada Standard. So the wording would be:

"Returns True if the reserved word **limited** appears in the Element."

Pascal wonders if Has_Private returns True for a private extension? It does include the reserved word **private**. Tucker thinks that these need to apply to things that have *optional* syntax.

Steve suggests that these queries should not include anything that is determinable by the element kind.

Someone suggests deciding based on the actual syntax in the Ada standard. Pascal prefers that we not depend on accidental writing of the syntax productions in the Ada standard; there are many different ways that the grammar could have been written.

Tucker thinks we have to modify Steve's rule for things that are commonly used together (tagged and untagged records), we should have a query for those cases.

Looking at private extensions, Tucker says that once you have Has_Private, if an element has the word **private**, then it should return True.

Should we have a function Has_Tagged? It's not required (you can tell from the kind). Tucker thinks it's unneeded. Jean-Pierre says that it is cheap to implement, and it seems useful.

All of these functions need lists of expected elements.

Brad asks why we don't have Has_Task and Has_Protected for interfaces. Has_Synchronized is really for private extensions. Surely this and Has_Limited will work on interfaces, it is weird to have two out of four possibilities covered. Add these functions.

Has_Protected also would work on access protected procedures.

Perhaps we could get rid of all of the separate access protected kinds. Or maybe just make subranges of these sorts of kinds. Randy should add An_Anonymous_Access_to_Object_Definition and An_Anonymous_Access_to_Subprogram_Definition subtypes to SI99-0004-1. This would be consistent with 3.9.12. Drop the traits from SI99-0004-1.

Move the text currently in the summary section to the recommendation section (because it isn't a summary!) and add a short summary.

Approve intent of this SI: 9-0-1.

SI99-0023-1/01 Usages of subtypes Name and Name_List in the ASIS specifications

The issue is whether is acceptable to change the formal parameter names. We asked the ASIS community, and there was a weak concern that this is an unnecessary change without a lot of value. Thus, Pascal suggests that we just change the function results.

Jean-Pierre suggests that we change the subtype of parameters, and leave the parameter names alone. Pascal and Tucker think that is too ugly to do.

Sergey really would want to get rid of the subtypes completely. That is out of bounds, it would be highly incompatible. He then says this change is completely useless and no one wants it. But Jean-Pierre (who does a lot of ASIS programming) says that *he* wants this, he depends on these subtypes for documentation. Sergey's contention is thus proved to be false; he should have said hardly anyone wants this. ☺

Pascal thinks that a note might be appropriate for these functions, as they could be mistakenly thought to work on an expression.

Straw poll for changing subtypes of parameters from Expression to Name (when the formal parameter is still Expression): 4-5-4. That was conclusive ☺. We'll leave it as it is.

The wording is empty so we can't approve this now.

Approve intent of SI: 9-1-1. Erhard would like to change *all* of the parameters (both the name and subtype).

Pascal still has the action item to update this SI.

SI99-0024-1/05 Provide a semantic model in addition to existing syntactic model

Starting at the top. There is a large enumeration and subtypes. Pascal comments that we should eliminate any abbreviations (that is, Subp and Enum), since abbreviations are inconsistent with the existing ASIS.

Declared_Views are things with names.

Ed asks if a prefixed view includes tasks and protected objects. Yes, those should be considered prefixed views.

The last three are connectors between the syntactic and semantic domains; an expression can be anything as it includes Name.

Sergey comments that he wants this to be a tagged type hierarchy since you will need to deal with heterogeneous lists (like View'Class).

Pascal does not understand why Tucker is worried about dangling pointers. It should be as easy as possible for users to work with. There shouldn’t be any need for anyone to worry about how memory is managed; that’s for the ASIS implementer to get right.

Should we have strong typing here? It’s surely necessary to process heterogeneous lists (which is View in Tucker’s model).

Jean-Pierre comments that he thinks that you will usually get an exception at runtime with either model. And a lot of conversions will be necessary, which they won’t gain anything. Sergey seems to agree. Tucker and Pascal strongly disagree.

Sergey complains that there are multiple levels of strong typing. ASIS generally depends on dynamic checking; he would like to flatten out this hierarchy.

Erhard says they have done projects with ASIS and with typed interfaces. The types make it easier to understand the interface as compared to the untyped model of ASIS.

Straw poll on static strong typing: 10-1-2.

What are the connections between syntax and semantic worlds? They’re at the end of the specifications.

How does Corresponding_View work? It returns what the expression denotes (“expression” might be a Name). For A + B, that is the function "+". It does not return the type of an expression or anything like that.

Erhard wonders whether the function for "+" exists in the symbol table. Some of these items will need to be created on the fly. The basic idea is that every item that exists in the semantics of Ada has a representation in this interface.

Steve comments that there isn’t a way to talk about uses; this only really returns declared items.

If you came from the syntactic domain, you surely should be able to get back. But when you come from something (like a generic instance) that doesn’t necessarily exist, you may not be able to get back.

What does Expression_Denoting_View do? Does it point at the declaration or use?

Pascal wonders why Primitive_Subprograms returns an ASIS.Declarative_Item_List? These are likely to be implicit, so we want to stay in the semantic domain here. It is not necessarily the case that the inherited routines have syntax, so this might not be useful. Moreover, we don’t want to have to go back and forth.

Returning to the tagged vs. discriminants discussion.

Tucker says that he thinks it is important to be able to create single objects that aren’t initialized and don’t require memory management. And that is why he wanted untagged types.

Pascal says that it is really a pity that we’re not using tagged types. After all, they do everything “right” (equality, composition, prefix calls, etc.). Tucker claims that the predefined equality will be fine here, as long as the untagged type is just a wrapper around a tagged type. But that seems that to be the worst of both worlds; if you have to use tagged types anyway, they ought to be visible to get the other advantages.

Tucker points out that the root operations are inherited in all of the subpackages.

Pascal and Randy wonder it there would be very much use of classwide operations. Most of the time, you would just be dealing with a subtype or a subprogram, or with lists of entities. List can be handled easily with Ada.Containers.Indefinite_Doubly_Linked_Lists. Someone complains about the implicit allocation in that container, but that is likely to be irrelevant: ASIS programs are not going to be running as hard real-time embedded applications!

Pascal also notes that it would be easier to understand if this was a tagged hierarchy.

Sergey wonders where compilation units are handled. They are handled in declared views. There are things that can't easily be determined from the syntax elements.

Sergey suggests that we start with entities rather than views. Ed comments that that matches the way compilers typically work. Tucker disagrees, private types for instance can be very different, and might in fact be represented separately.

Pascal says that the language is defined in terms of views. Ed comments that what they get in their implementation usually is the current view of an entity, not just an entity.

Jean-Pierre wonders if there is a possibility to finding a particular view of an entity from a particular element. Tucker originally had something like that, but it was killed. We're not trying to make it possible to write a debugger in ASIS. Jean-Pierre wonders about the "current" element; there is no such thing. You can't do name resolution for an arbitrary identifier at a specific location; that is too hard.

There is a routine to determine whether two views represent the same entity.

Is there a way to get all of the views of an entity? No, there may be a large number of those (including renaming, which have different parameters, etc.).

Tucker says that he will try to make a tagged version of the interface, and then we'll need to look at potential uses. Tucker will try to create some examples using their static analyzer for both versions of the interface. Pascal suggests trying to use limited with clauses in the tagged version to try to flatten it out.

Ed says that this would make it possible to make more tools in ASIS, because it would provide additional information.

Perhaps we need a singleton container for storage management of View'Class. (But a definite singleton container isn't very interesting!).

SI99-0025-1/01 ASIS 99 allows too much variability between implementations

There is a discussion of what current implementations do. In particular, which of these allowed variabilities are actually used? If they are not used in practice, we should consider getting rid of them in order to make ASIS applications more portable.

What implementations are there? We know about the AdaCore and IBM implementations. Greg notes that AONIX has an implementation, too. Jean-Pierre says that he tried to use it and gave up as it is a partial implementation and is fairly restricted in what it supports — not enough for his applications. Most of the Corresponding_xxx routines return Nil_Element unconditionally. As such, we'll look mainly at the AdaCore and IBM implementations (anyone can make a partial implementation without any special permission from the standard).

Why do we want to change this? There is not much point of having things optional in the standard; we don't need a standard if we're going to let implementers do whatever they think is easiest! We want the standard to be useful to users, and we want claims of conformance to mean something.

Most of these "options" cannot be worked around. That is, if the implementation doesn't provide the information, there is no other practical way for an ASIS application to get it. So what value is there to allowing these options? It isn't the case that an ASIS program can test the flags and do something different; at best it can abort. We conclude that an implementation that does not support most of these things is not complete — and thus should not be allowed to claim to be a complete ASIS implementation.

The normalized cases surely make it impossible to recover source; things like default expressions could be very wrong. Jean-Pierre makes the point that both normalized and unnormalized forms are useful, and that is selectable on the calls to the various operations. There is no reason for it to be a global capability only providing a single form.

So all of the functions with Normalized in their names in 7.xx should return False — we don't want to allow this variability. Is_Formal_Parameter_Named_Notation_Supported should return True, and Default_In_Supported should return True.

Tucker wonders about comments. Should they be required? Should line numbers and columns be required? It seems so; no one can think of a good reason that they shouldn't be required, and surely text processing applications need them (especially comments). These should be supported and the associated functions in 7.xx should always return True.

We turn to Attributes_Are_Supported. What attributes are being discussed here? In ASIS, *attributes* are relatively undefined compilation unit properties. They are not portable in any case; since they are implementation defined. A list of supported attributes would be more useful than this function, but we're not sure anyone cares. Leave this as is.

Implicit_Components_Supported seems to be unnecessary; but someone objects that you need to use this to find the total size/layout of a record. Someone else objects that you should use Data_Decomposition for this. But that's optional. Usually Record_Components (16.30), Implicit_Inherited_Declarations (16.4), and Implicit_Components (16.31) are needed to get all of the components. (Not the best design.) This permission covers implementation-defined components ("implicit" is a bad name). 16.31 returns Nil if Implicit_Components_Supported is False. Pascal says that this might be determined fairly late in compilers. Requiring this might cause a burden. Leave this alone.

Predefined_Operations_Supported goes with Corresponding_Type_Operators (16.1).

Jean-Pierre argues that tools that are analyzing uses of operators currently hit Nil, and that causes all kinds of trouble. Randy argues that this belongs to the semantic interfaces; Corresponding_Type_Operators should be obsolescent and thus we don't care about this function. The lists of primitive operations returned by the semantic interface will necessarily include these along with all inherited operations, so worrying about the syntactic side of things is not necessary. One way or the other, the predefined operations need to be available.

GNAT does not support this. We're going to require everything that GNAT and IBM support; this is the only major difference. So the question is whether than GNAT is willing to support this; if so it should be added to the standard. We can't decide on 7.15, leave it for now.

Inherited_Declarations_Supported (7.16) and Inherited_Subprograms_Supported (7.17) will return True; they are supported in both major implementations.

Generic_Macro_Expansion (7.18) also should be True, a non-nil result is required from Corresponding_Body. The ASIS users present say that it is too important to not require expansion.

This doesn't require macro expansion for generated code, just for ASIS purposes.

The SI author should also change the places that used to allow returning Nil; with some of the permissions above removed, a number of functions won't return Nil anymore.

Approve intent of SI: 10-0-1.

SI99-0026-1/01 Baseline version of ASIS standard

In item (4), change "description was" to "descriptions were".

In item (6), change "keywords" to "reserved words".

In item (9), change "are" to "is".

Greg objects to item (7). Randy explains that the tools generate cross-references by clause name; thus they have to be unique. This was inherited from the Ada 95 tools (Scribe), and is very fundamental to the design of the tools; it wouldn’t make sense to change it. He does agree that the error messages could be better (that is, there ought to be some, rather than just using the wrong clause number for duplicate uses). Pascal notes that he thinks the unique names are an improvement, otherwise you have subclauses like 10.15 and 10.16 that have the same name. That makes it harder to use the Table of Contents. We agree that item (7) is OK.

We don’t need a review now, as we’ll have to do a careful review of all of the text in the Standard at the end of the process.

Approve SI with changes: 10-1-0. Bibb voted against, as he would like a formal review of the changes before approving them.

SI99-0027-1/01 Obsolescent features should have their own annex

Obsolescent features are not moved from their packages; only the text will move.

This particular item is listed as obsolescent in the old standard; we want to move it to this new Annex. There’s no other obvious place to do that. For other items, their own SI will mark them as moved here.

Drop the first sentence of F.2.2, and move the second sentence to last. “functionality” should be “function”.

Erhard worries about the redundancy of “the use is not recommended” in each entity. Pascal thinks that is useful when someone find one of these obsolescent entities directly via search or the table of contents. Erhard agrees.

Approve SI with changes: 8-0-2.

SI99-0028-1/01 What does appropriate kinds mean?

The Status for the raised exception should be Value_Error, and that should be added to the statement of “Raises...”:

“Raises ASIS_Inappropriate_Element with a Status of Value_Error for...”

Approve SI with changes: 9-0-2.

Greg will actually do this change.

SI99-0029-1/01 Inconsistent inconsistent list

Randy explains that 12.1 specifies that there are three pairs of units returned and later that there are six pairs of units returned for the inconsistent list. One of these must be wrong.

What is a “supporter”? After much confusion, we find the term is defined in 3.12.4. The new definitions section would have helped here.

GNAT’s implementation cannot get inconsistent units, because it requires compilable units. It probably can happen for Rational. Tucker says that their implementation would not allow units to be inconsistent. Randy says that it probably could happen for their compilers and their hypothetical ASIS implementation.

Pascal reads the comments in IBM’s ASIS source. There is a minimal or maximal form for these lists (the maximal form is the closure of the compilation units). The examples show the minimal and maximal forms, but don’t seem to explain when either is required.

We agree some rewording is necessary. Pascal gets the short stick, er, action item to fix this wording. Bill might have ASIS 83 info, he’ll provide that to Pascal.

Jean-Pierre's request to find the Size of an object

Randy notes that Jean-Pierre had sent a request to the ARG list asking about a mechanism to find the size of an object. Jean-Pierre says that he thought the answer of "use the Data_Decomposition package" was definitive. Randy and Pascal were not so sure; that package is optional.

We turn to look at the Data_Decomposition package.

Steve Baird wonders how 22.20 (Discriminant_Components) would work on task and protected types that have discriminants. Data_Decomposition is only about reading and writing; that doesn't make sense for limited types. In any case, this is irrelevant to the question at hand.

Several people are confused by the purpose of this package. Data_Composition is an ASIS 83 thing that really is covered by the streams operations in the Ada standard. No one recalls anyone actually using it.

The Size function (defined in 22.31) seems to be requesting the size of a stream for the object, not the actual size of the object. (Remember that 'Stream_Size is not the same as 'Size.)

There is an action item on Steve to check on whether this works for protected and task types. But he would rather let Data_Decomposition decompose. Sorry, but now that he brought it up, it needs to be analyzed for problems.

If Jean-Pierre wants to be able to find the size of an object, he should propose an interface for that, either as part of the semantic interface or separately. Data_Decomposition seems inappropriate for the task.

Detailed Review of Ada 2005 regular AIs

AI05-0002-1/02 Unconstrained arrays and C interfacing

Steve says that he made the changes requested in Albuquerque.

Pascal notes that the summary is wrong; this now only applies to C and related conventions. The AI also adds correspondence for function results.

We look at the new wording paragraph added after B.3(71). What does "returned as by" mean? "returned in the same way as" is the meaning. We try various rewordings: "as is returned by"; "returned in the same as by a C function"; "returned as would be a result of a C function". Nobody is very happy with these.

Jean-Pierre suggests, "The rules of correspondence given above for parameters of mode **in** also apply to the results of functions." "results" isn't well-defined. Could use "return object" instead.

"The rules of correspondence given above for parameters of mode **in** also apply to the return object of a function."

Pascal suggests making 71.1 into a bullet, then we don't need extra text. Move the new paragraph after it (also as a bullet).

Reorder wording paragraphs in AI so 62 comes before 71.1.

No "or" in the bullets for 62. Reorder so the first bullet goes last.

Tucker wonders if there is anything that implies that interfacing pragmas are supported for objects with convention C. We find that in B.1, not in B.3. Should there be any correspondence rules for objects of convention C? Such objects don't seem to be used enough to worry about portability; C interfaces are almost always sets of functions.

Drop the leading commas in the discussion section.

Approve AI with changes: 11-0-0.

AI05-0008-1/04 General access values that might designate constrained objects

We approved this last time, but we had a mistake. Randy tries to explain the convoluted way the problem was discovered, and how this was fixed.

Tucker tries to figure out why this was needed at all; we look at the Porto minutes and note that 3.7.2(4) does not cover the discriminants, only discriminant-dependent components. This seems to be everyone's reaction when they first see this AI.

Approve AI: 6-0-4.

AI05-0009-1/03 Confirming rep. clauses and independence

Pascal would like to drop "other than Ada", because usually the convention Ada is confirming. Thus it is covered by the confirming representation clause exemption.

Jean-Pierre asks if, in the absence of any clauses, you have independent addressability. Yes, of course.

Steve asks about a case where a type puts two record components in a single byte, then a derived type confirms that. The new type should also not be independently addressable. So just saying "non-confirming" is not quite enough.

Pascal suggests that the wording should say that a confirming representation item never changes independent addressability, and a non-confirming representation might lose independent addressability in an unspecified manner.

Tucker notes that the part about the composite object got dropped; we still need that text.

"Normally, any two nonoverlapping objects are independently addressable. However, if a non-confirming representation item is used to specify packing, record layout, Component_Size, or a convention for a given composite object, then it is unspecified whether or not two nonoverlapping parts of that composite object are independently addressable."

Tucker worries that we have a possibly broken definition of "confirming" (based on Steve's example). Steve suggests that possibly we should allow both representation clauses to co-exist. Randy suggests that confirming a non-confirming representation clause is non-confirming.

Tucker now suggests that we should define "non-confirming representation clause applies to...".

Tucker tries to do wording, and a bunch of unrelated discussions erupt.

Steve notes that 13.2(9) has a case where a non-confirming representation clause makes a difference. Delete "Component_Size is specified" from 13.2(9). Is is suggested that Bibb add this to AI05-0012, but since that AI was tabled later in the meeting, it should be added to this AI.

Tucker's wording is ready. Change the last line of 13.1(15/1) to: "An inherited aspect of representation is overridden by a subsequent representation item that specifies {a different value for} the same aspect of the type or subtype."

Steve is sold. In Steve's example, both aspects apply (because a confirming clause does not override an earlier one), and the first is still non-confirming. (Better have an AARM note that explains this!) Steve notes that we don't care that there exists a confirming clause if there is a non-confirming clause.

Pragma Independent should be usable inside a record definition to apply to components. And then it only needs one parameter. The name resolution rules should reflect that and be similar to Atomic. We don't want to apply Independent to a record type, it would be confusing. We could have Independent_Components apply to a record type. In fact, the minutes of the Albuquerque meeting indicate that we decided that last time. (Specifically, that Independent applies to components, and Independent_Components applies to any composite type.) What is the name of the argument name then? Record_or_array_local_name is suggested. [Editor's note: but this is not ideal if the pragma applies to any composite type.] Should this apply to anonymous array objects? But we surely don't want to handle this for objects

“It is illegal to apply a pragma Independent to a component that is not independently addressable from all other components of the enclosing type. It is illegal to apply pragma Independent_Components to a type unless all components are independently addressable from one another.”

“If a pragma Independent or Independent_Components applies to a generic formal type, then the actual type shall have independent components.” We’d like to get rid of this notion of independent components. Tucker thinks this should work in the default case, as well as when the pragma is given. Bibb suggests just saying if the components are independently addressable. That would have legality rules depend on unspecified behavior. That seems bad.

Tucker wonders why this is unspecified. We could just say it is not independently addressable if there is a non-confirming representation clause. Pascal says that defining a term that is non-intuitive is not helpful.

We should just require matching of the pragma for formal types.

Should **aliased** components be required to be independently addressable? Yes, add something to 9.10(1) to cover that.

Drop the static semantics and the term "independent components" from C.6.

Pascal asks if address clauses could make top-level objects non-independent. Tucker says that is not allowed by 9.10(1). [Editor’s note: Tuck’s argument is unclear here: the address value in an address clause is only known at run-time, so how could an implementation prevent address clauses from compromising independence? Restricting the addresses allowed because they *might* compromise independence in a rare instance would idiotic.]

Approve intent: 7-0-3.

Note that the discussion of AI05-0012-1 suggests that a rule that defines that Atomic objects are independent of all other objects (including components) be added to 9.10(1).

AI05-0012-1/02 Independence and Representation Clauses for atomic objects

Bibb tried to explain what he did.

Volatile does not require independent addressability; it should not be included in 13.2(6.1/2).

Tucker doesn’t think we need independence in C.6(10-11). He claims that indivisible implies independent. Jean-Pierre says indivisible is in time, independent is in space.

Tucker suggests that we should add to 9.10(1) some text that defines that Atomic objects are independent of all other objects (including components). That would be preferable to changing C.6(10-11). Probably should do this in AI05-0009-1.

13.2(9) is confused, it seems to be circular with 13.2(6.1/2). The idea is to reject the combination of Pack and Atomic_Components on a packed array of Boolean. But that is too special of a case to write a rule for.

Tucker suggests that we change 13.2(6.1/2) to an implementation permission.

Randy and Tucker get into another long winded and heated argument on the meaning of Pack where they get no agreement.

The problem remains; implementations reject/ignore this combination of pragmas, and that means that they don’t conform to Annex C given the current wording of the recommended level of support for Pack. That needs to be fixed somehow.

We decide that we just cannot get agreement now, so we will just table this AI. There is a real problem, but we can’t agree on a solution, so we’ll leave it open for now.

AI05-0017-1/02 Freezing and incomplete types

Pascal has trouble understanding what he wrote.

Tucker wonders why stubs need to be mentioned here. Pascal says that in the example, the intent is that T1 is frozen after the stub of P1.

We discuss (again) why we need freezing for simple incomplete types. That is package STT (in the e-mail of August 3rd). This example could be written in the example of Pascal's original question. Tucker says no, that is not possible, because you cannot have visibility on the private part. Tucker again wants to remove stubs from this rule. Pascal still wants stubs to be mentioned here, because otherwise the replacement of a proper body with a stub would change the freezing properties.

Tucker wants the full type to freeze the incomplete type; there is only a problem if there is a body between the incomplete type and the body containing the full type declaration.

Ed points out the end of the declarative part of the body will cause freezing, so there isn't a need for a special rule about freezing at the end of a body. That's true for the completion of an incomplete view (freezing of which will necessarily freeze the incomplete view), but it is not true for the incomplete view itself if there is no completion. Randy wonders if freezing is necessary if the completion is missing; others say that a missing completion is illegal anyway by other rules. So the first insertion is unnecessary, drop it completely.

We turn to the second insertion. Tucker claims that the second part (which is describing "normal" incomplete types) is unnecessary. He says that if you are after the full type, the incomplete type (view) isn't important; that means that only "in the immediate scope" matters. After reflection, no one can find a problem with that, so drop "or if" part of the second insertion.

Tucker would like to add to the discussion section: "The reason why it is important to freeze incomplete types is to prevent premature uses in calls. For example: (give Pascal's STT example)."

Replace the last paragraph of the discussion section with "Note that the start of the package body P1 does not cause freezing of T1, but the end of the declarative part of the package body will freeze everything including T1."

Approve AI with changes: 7-0-3.

AI05-0018-1/02 Formal package matching rules

Steve claims that the RM says that the two values are not equal, and he changed the matching rules to make that work. Erhard and Ed wonder why this doesn't occur in many other cases? For instance, in static matching of range constraints. Steve claims that the problem can only occur when the types are different.

Erhard complains that 3.14 and 3.14000000000000001 would match using this rule. That's ugly.

Jean-Pierre complains that the legality of a program would depend on your target. Pascal says that is already the case, for instance if you use a float compare as a choice in a case statement.

Tucker wonders why his proposal (from last time) was rejected (that is, the value is always exact until used). Steve says that it would be an earthquake in some implementations (his).

Randy agrees with Tucker. Tucker says that a named constant of a type would be rounded (because this is effectively an assignment); but other cases would not be rounded until use.

Pascal says that they round right away, so it would be an earthquake to fix. Tucker thinks this is a single special case, and should be fixed.

Erhard says that he doesn't want a large paragraph in the manual to fix this Baird pathology. Pascal says that he is willing to sweep this under the rug.

Tucker suggests that we could just say "prior to rounding or truncation". Pascal would prefer to vote it No Action as just a pathology.

Tucker suggests putting these minutes into the !appendix of this AI.

Reclassify this as pathology, and vote it No Action: 8-1-1. Bibb opposes since he is not convinced; while this case is a pathology, he worries that there are other related cases which are not.

AI05-0019-1/02 Primitive subprograms are frozen with a tagged type

Pascal explains that we now separate freezing subprograms and freezing the subprogram's profile. The latter is frozen only for calls.

Tucker worries that some link names depend on the profile. Pascal and Randy think this issue is fixable with work. It might be necessary to delay generation of the link names until the profile is frozen. Surely, they won't be used until the program is linked, and the profile must be frozen by then.

Ed notes that the example in !discussion is messed up. Change second parameter type to T2'Class. **Tagged** and **with** need to be swapped between the two full declarations of T1 and T2.

Pascal also added a rule to freeze the subprograms mentioned in attribute definition clauses for a type.

AARM Note 13.14(15.1/2) should say "The second sentence is the rule...".

Remove the author's note.

Approve AI with changes: 7-0-3.

AI05-0024-1/03 Run-time accessibility checks

This was the reason that anonymous access parameters are not allowed for entries. Disallowing class-wide parameters for entries seems like too much, however.

Tucker goes through the changes sequentially.

Put the changes to 3.10.2(14.-3) first (they should be in order).

3.10.2(14.4/2) also has to be changed to say "in the first case" instead of "in this last case".

The TBD should be deleted, we prefer the given wording rather than the alternative.

Steve wonders if:

```
task body TT is
begin

   accept E(X : T'Class) do

      declare
         type Acc_TC is access T'Class;
         P : Acc_TC;
      begin
         P := new T'Class'(X);
      end;

   end E;

...
```

would pass the check. Yes, the check is fairly complex (we surely don't want it to fail all the time!).

4.8(10.1/2) should have the insertions and deletions marked.

What about implementation cost? It’s possible, but not necessarily cheap. This is only going to happen in accept statements, so it’s possible to avoid overhead in other cases. Tucker’s anchor solution could be piggybacked on existing finalization or other existing mechanisms.

Approve AI with changes: 4-0-6.

AI05-0028-1/04 Problems with preelaboration

Tucker says that “entities” is wrong in 10.2.1(11.1/2), because that makes the full type have P_I, which is not what we want. Tucker will take an action item to check and rewrite this.

For Q 7, the wording is a bit confusing. Replace it by:

“The creation of an object (including a component) {that is initialized by default}, if its type does not have preelaborable initialization.”

Approve intent of AI: 10-0-0.

Later in the meeting, Tucker describes what is wrong with 10.2.1(11.1/2). He claims that the definition is circular, in particular for protected types.

Anyway, after Tucker talks for a long time (this was the last thing we did at the meeting, and the editor was short on of energy for note taking), he gives his wonderful new wording for 10.2.1(11.7/2):

“If the pragma appears in the first list of basic_declarative_items of a package_specification, then the direct_name shall denote the first subtype of a [private type, private extension, or protected type that is not an interface type and is without entry_declarations] {composite type}, and the type shall be declared immediately within the same package as the pragma. If the pragma is applied to a private type or a private extension, the full view of the type shall have preelaborable initialization. If the pragma is applied to a protected type, {the protected type shall not have entries, and} each component of the protected type shall have preelaborable initialization. {For any other composite type, the type shall have preelaborable initialization.} In addition to the places where Legality Rules normally apply, these rules apply also in the private part of an instance of a generic unit.”

Ed says that the last sentence 10.2.1(11.2/2) should say “has”.

Approve AI with changes: 10-0-0.

AI05-0034-1/01 Categorization of limited views

Bibb wonders if there would be any future problem with this. No, there is never any elaboration for a limited view; that invariant is not going to change (it’s a significant part of the point of limited with).

Approve intent: 9-0-1.

Randy needs to provide wording for this.

AI05-0035-1/02 Inconsistencies with pure units

Looking at question 1. Change “checks” to “rules” in the question.

The ancestor type must already have the appropriate properties, and you can’t repeat it.

Moving on to question 2. The wording is not well liked. But this is very similar to the wording used for Preelaborate. Pascal thinks that we still need to “hook this up to” preelaborate. No, Pure units are also preelaborable and so that is already covered.

On to question 3: No changes needed.

Looking at question 4. The wording has a bracket that is backwards; the one following “then its” should be a closing bracket rather than an opening one.

Tucker worries that 15.3 needs some wording to apply to formal types. Randy notes that formal access types can’t cause trouble, but Tucker is worried about formal private types. Does “partial view” include formals in an instance? Add to the end of the new paragraph following 15.5/2 “...presuming any composite formal types have non-visible components whose default initialization evaluates an allocator of an access-to-variable type.”;

Approve AI with changes: 8-0-2.

AI05-0037-1/01 Out of range <> associations in array aggregates

Randy explains the problem, which is essentially that <> is not an expression, and we need this rule to apply to both. He checked the entire clause for additional occurrences of this problem, but didn’t find any.

Approve AI: 10-0-0.

AI05-0040-1/01 Limited with clauses on descendants

In the question, “ocver” should be “cover”. Also in the question, “this rules doesn’t apply” should be “this rule doesn’t apply”.

Tucker would prefer the summary “A limited with clause may not name an ancestor.”

Change the wording to: “in the context_clause for the explicit declaration of the named library package {or any of its descendants};”

Approve AI with changes: 10-0-0.

AI05-0042-1/00 Overriding versus implemented-by

Pascal will take this one.

AI05-0043-1/00 The Exception_Message for failed language-defined checks

Tucker notes that the version number for this one should be /01 (this is fully written up). Then he moves to approve it.

Approve AI with change: 8-0-2.

AI05-0045-1/01 Termination of unactivated tasks

Steve and Randy explain this AI. The existing wording does not work for extended return statements, and also requires predicting the future.

Tucker proposes revising the wording to:

“At the start of this step, any tasks dependent on that master that have not initiated activation become terminated and are never activated.”

Erhard worries that this new wording requires dead bodies to stay around longer. Randy wonders if there is any case where that matters. Erhard tries very hard and then comes up with an allocator that gets aborted; the task’s master would be an access type at a more global level. Randy complains that storage management of that case is not well-defined anyway (the rules imply that the storage cannot be recovered until the master is finalized anyway).

Steve shows an example of why the AI wording is wrong:

```
      procedure P is

         type Ref is access T_has_Task;
         Ptr : Ref;
         task Child;

         task body Child is
         begin
            Ptr := new T_has_Task;
         end Child;

      begin
         null;
      end P;
```

If P starts waiting at precisely the instant that Ptr is allocated (but before it does any activations), the tasks being allocated are killed. For no reason: Child is still running. So we do have to write this in terms of creation/activation.

Tucker suggests replacing the wording with “If a master includes the creation of a task, and it begins finalization prior to either initiating the activation of the new task or returning the task to a caller as part of a return object, the new task is never activated and becomes terminated.”

This is a transitive rule in the sense that the outer master that is handed the return object is also one that “includes the creation of a task”. Thus a “normal” return of a task is properly handled if something prevents that from being activated.

Erhard has trouble reading that from the wording. Tucker suggests adding a remark to clarify that: “If a master includes (directly or indirectly) the creation of a task, and the master begins...”

This would replace 9.2(6). All other stuff in the AI goes away.

[Editor's Note: This wording is wrong, or at the very least the use of “includes” is unclear. The allocator in the example above surely is included (indirectly) in the master of P, and the scenario outlined above would result in the allocated task being killed while Child is still running. We'll need a different solution.]

Steve announces this is incompatible with coextensions (they’re not a “part”). Add “as part {or coextension} of a return object...”.

Do we want to see this AI again? Pascal thinks so.

Approve intent of AI: 12-0-0.

AI05-0046-1/01 Null exclusions must match for profiles to be fully conformant

This is only for full conformance, not subtype conformance.

Erhard complains that **in** is not required to match in this case. He thinks that we need a better argument for this change than the one given in the discussion section of the AI.

Removing the **tagged** here would make the full conformance fail. Similarly, this would be illegal if the subprogram was not primitive. Erhard thinks these are better arguments. Add this information to the discussion.

Drop “either” from the insertion: “both or neither...”.

John wonders why this doesn’t apply to subtype conformance. We allow any syntax that gets the exact same semantic result for subtype conformance.

Approve AI with changes: 10-0-0.

AI05-0050-1/01 Return permissions are not enough for build-in-place

Tucker says that his model is that build-in-place returns are like an **out** parameter; thus the constraint checks will be performed before the call. The permission of 6.5(24/2) is intended to allow that model. And thus should be fixed to handle case (1). Drop "If the result subtype of a function is unconstrained, and..." from that permission.

(2) means the *final* result is the one that the permission applies to, not any earlier one. We don't want the permission to raise exceptions that would not otherwise be raised. So, change the wording of the permission to "the final value of the return object" instead of "result" in 6.5(24).

The permissions should apply transitively. Pascal thinks that an AARM note to point out that the wording applies transitively would be enough. Steve agrees.

Approve intent of AI: 10-0-0.

Steve will write this up, adjusting the discussion for this intent.

AI05-0051-1/01 Accessibility of dispatching function calls (aka another Baird question)

Steve tries to explain the problem. There are issues if you make a dispatching call into a more nested scope for a function that returns something with accessibility (T'Class, discriminated record, anonymous access).

Tucker had suggested that you could pass in the level of the statically called function in order to properly make the appropriate check.

Steve's original problem is what storage pool to allocate from. That could also be handled by passing in the pool.

This only would need overhead for functions with one of these kinds of returns that are (or could be) dispatching.

The AI needs a question.

Steve also has a second question which should have been a separate AI. Split the AI.

He thinks that the result value (not the function result) needs this check, because there might be an access discriminant. This is not limited to limited types; that means that for *every* function that has a class-wide return, you have the overhead of this runtime check.

Tucker says in the example, you statically know that it has an access discriminant. But you could have had the local function return a class-wide object, and we don't want to look at function bodies.

Tucker thinks that this changes the model of Ada programs for a user. Pascal says that this is mysterious to the user; a Program_Error would come from somewhere with no obvious reason. Tucker would like to make something illegal; but exactly what isn't obvious.

Tucker starts thinking about rules. Randy says that Tucker needs to take this off-line because the rest of us can't help.

Tucker and Steve will work on both of these AIs.

AI05-0052-1/01 Coextensions and distributed overhead

Steve notes that adding a limited coextension to a non-limited type essentially allows tasks to be added to non-limited types. This is a distributed overhead on non-limited types. Steve says that a master and activation chain would have to be passed to functions returning any class-wide type.

Pascal says this is very bad performance hit for existing functions; that is unacceptable to him. He would like to make limited coextensions illegal for non-limited types. The allocator would be illegal if the enclosing type is not limited (access discriminants of limited types would be OK).

If it is not class-wide, there isn't much of a problem. So it could be allowed for untagged. Randy complains that might cause problems in shared generic bodies, as a non-limited private type could have tasks. For instance, assuming that Tsk is a type that contains a task:

```
generic
      type T is private;
package G ...

type Ugly (D : access Tsk) is record ...

package P is G (Ugly);
```

If the body of G does something that requires the destruction of a coextension, it would have to be aware that there are tasks. Others are dubious that there are any such cases. But this is trivial: just give the discriminant a default expression that is an allocator. Then an ordinary object declaration has all of these issues:

```
generic
   type T is private;
package G is
   procedure PP;
end G;

package body G is
   procedure PP is
      Obj : T;
   begin
      null;
   end PP; -- Might have to wait for a task here. Ugh!
...

type Ugly (D : access Tsk := new Tsk) is record ...

package P is G (Ugly);
```

[Editor's note: A similar effect also can occur with allocators of a local access type of T, components of T, and perhaps other uses of T. Ban coextensions! Ban them all!! Oops, got a bit carried away... ☺]

Pascal will take this AI and make an effort to deal with this., possibly combining it with the AI split from AI05-0051-1.

AI05-0053-1/01 Aliased views of unaliased objects

Should we ban aliased return objects? Surely no one uses them. There are obviously a number of problems with them.

The recommendation is to remove **aliased** from the syntax. Pascal notes that 3.10 defines which objects are aliased, and those don't include return objects anyway. So this syntax doesn't actually mean anything. So there is actually no problem except the confusion provided by having an unused keyword.

Someone notes that this is the second time that we change this particular syntax. So this is just a minor nit: we wrote **aliased** when we actually meant **constant**. ☺

Approve intent of AI: 10-0-0.

Steve will need to rewrite the discussion of the AI to reflect that there are no aliased return objects, but explain why they would be dangerous if they existed.

AI05-0054-1/01 Variable views of constant objects

Steve comments that the Rosen trick is erroneous by 13.9.1(13), which Jean-Pierre finds surprising.

Pascal does not like that constant does not mean constant. Up until now, declared constants were constant between the initialization and finalization.

Tucker would prefer to make this legal. Ada 95 has constant views of variable limited objects.

Pascal wants **constant** to be believable for the user.

Tucker says that it is completely obvious to the user of the type that this will happen.

Randy complains that a runtime check is not acceptable. Erhard agrees. A private type could not be declared as **constant**, because a Program_Error might be propagated. The client would have no way to know that this might happen, and wouldn't care how a well-written abstraction was created. It also would prevent the writer of the abstraction from ever adding a self-referential pointer, because someone *might* have declared a constant somewhere. That seems like a major abstraction violation.

Should this be no action? Pascal objects, if it *is* useful, we should at least fix the erroneous rule.

Jean-Pierre says that it is not erroneous in Ada 95, the wording says "constant object" rather than "constant view", and in Ada 95, it would have to be a constant view of a variable object. So there is no problem.

Randy had suggested that we just add "nonlimited" to 13.9.1(13). "nonlimited constant object".

Tucker would like to restrict the change to self-referencing types. But that's hard to define.

Pascal suggests (as Randy originally did in e-mail) that we simply say that a limited constant is a variable view. Randy still worries that might change something else.

Coding conventions might require users to put **constant** on objects, they are not going to break privacy to check if this is OK.

John wonders if there is a way to statically make this illegal. Others say that has problems; any such rule would have to be privacy breaking (for instance, making declaring constants of types that contain self-references illegal), or incompatible (making some self-references illegal). There is one rule that isn't either: ban constants of limited types, reverting to Ada 95. But we don't want to go there. And static checks could not detect all cases, such as a self-reference created during an Initialize routine.

Pascal points out that there are variable views of constant objects (i.e. during initialization).

Tucker wants to list the options:

(1) Change the erroneous wording to allow this;

- (a) for limited;
- (b) for possible self-reference or controlled. (The latter so that Initialize and Finalize aren't erroneous.);
- (c) dereferencing a variable view created from a constant view.

(2) Admit existence of variable views of controlled and limited constant objects (add a user note);

(3) Raise Program_Error

- (a) At declare of constant of type with a self-reference.

(4) Make it statically illegal (but we think that is impossible without making virtually everything illegal).

(1) and (2) are really the same thing, but have different effects on the wording.

Steve shows an example of another way to get a variable of a constant view. It uses extended returns and an Unchecked_Access to get a variable reference of a constant view. But his example is occurring during the initialization phase, when this is still a variable view. Initialize also could do this (and without such complications).

Steve wonders about aggregates. Randy says that these are always built in place for limited types.

Straw poll: Nothing wrong (cases (1) and (2)): 5; catch at runtime (case (3)): 3; Hate both: 2. That doesn't help move us forward.

Bibb moves to keep alive and discuss. Pascal and Tucker will write up the two alternatives, with wording and rationale. And we'll discuss them again. Yuck.

AI05-0055-1/01 Glitch in EDF protocol

The !discussion header is missing, should go before "The current rule...".

Alan explains the problem in detail. (See the paper in http://www.ada-auth.org/ai-files/grab_bag/EDFpaper.pdf for this explanation.)

Tucker worries that the wording is confusing. Alan also notes a buglet in the wording. Tucker suggests "; and furthermore, ...". He'll check that during the break.

Typo: "papar" in example.

After break, replace the wording with "; and furthermore T has an earlier deadline than all other tasks on ready queues with priorities in the given EDF_Across_Priorities range that are strictly less than P."

Approve AI with changes: 10-0-2.

Alan asks the editor to send him an e-mail with the AI text once it is finished.

AI05-0056-1/01 Wrong result for Index function

Randy explains the fix.

Tucker thinks that we're fixing the wrong bug: he thinks that new function also should return 0 for a null string. Then the existing equivalence rule is correct.

Put in front of "If From...": "If Source is the null string, Index returns 0; otherwise,". For both versions of Index.

This seems to change the semantics in the case of a null pattern (which used to raise Pattern_Error). We think most implementations check the pattern first, the original wording surely is not clear about the order of those checks. Best to leave well-enough alone for that. (Randy wonders what the ACATS does, in particular, he wonders if it checks whether an exception is raised by Index ("",""), and which exception is raised by Index ("A", "", 2))

Approve AI with changes: 8-0-2.

AI05-0057-1/00 The class attribute of a constrained subtype

Assigned to Steve Baird.

Detailed Review of Ada 2005 amendment AIs

AI05-0030-1/01 Requeue on synchronized interfaces

Alan explains that the IRTAW wants this facility, but they do not care much about language completion — that is, they don't care about what happens in corner cases such as if it is a real procedure.

We move into a discussion of whether we should have primitive entries. That would solve the problem. Tucker thinks we should not consider upward incompatible changes (that is, we shouldn't consider making timed entry calls illegal for primitive interface procedures).

A pragma is suggested. Randy and Pascal think that violates good taste in pragmas. Tucker says that this would be a suggestion; requeue would raise Program_Error for a procedure.

Steve Baird says that such a pragma is not checkable semantically. Consider an interface types with no pragmas. Then you have a formal interface type derived from that type, which is abstract. The pragma would be at the point where you extend. Tucker points out you can't extend in a generic body.

Pascal suggests using pragma Convention (Entry) for this. "entry" is a reserved word.

Pascal would rather that this actually works, rather than raises Program_Error. Protected procedures are relatively easy, internal protected procedures are complicated trouble, and normal procedure requires leaving a protected action. Alan thinks that semantics would screw up implementations.

Timed entries already do this; Pascal says that we're already in a deep pit. Maybe, but requeue is a bottomless pit (with sharp sticks at the bottom, according to Ed).

Pascal worries that once in a blue moon you'll get a Program_Error; he wants a real check of some sort. Erhard points out that for the implementer of a class, there is no way to know that this procedure is used in a requeue.

Steve suggests Program_Error gets raised all the time for requeue without the pragma. As long as there is minimal coverage, that would eliminate any safety issue.

Tucker asks that we decide whether we want to do this at all. Pascal notes that his reaction to Alan's paper was that perhaps we ought to wait a few years to see if this comes up in practice.

Ed says that seems to be a natural extension, but it was a lot of work. To date, he has no examples other than Alan's. So this doesn't seem important right now.

What about the implementation cost? Ed says that it seems like a lot of work. Others note that there is very little likelihood that this is going to fall out cheaply (as these things look like procedures, and finding the entry information is very hard).

What is the workaround? Essentially, don't use interfaces in this case. That doesn't seem like a good thing, so we seem to be between a rock and a hard place (and hovering over a deep pit with sharp sticks at the bottom).

Pascal suggests that requeue on a procedure is a bounded error, it either raises Program_Error or otherwise it works properly. Properly means that a protected procedure would work like an entry with barrier True; a regular procedure would just be a call. Alan should make the bounds of the error as broad as possible.

Separate the pragma/syntax to specify entries into a separate AI; that would increase safety but the use of a pragma is contentious. And it can be added in the future presuming that we're only looking at compatible semantics.

Approve intent of AI: 10-0-2.

AI05-0049-1/00 Extend file name processing in Ada.Directories

Tucker says that the user could fix this if they wanted; they don't need to wait for a standard. Even so, he says that he is in favor of a fix. Randy worries about compatibility if we change the behavior of the existing routines.

Pascal asks the meta-question of what we want to do when we find missing capability in the language-defined packages. Should we forget it, or add new capabilities as a child, or add it directly into a package. Jean-Pierre says that making something a child *just* to avoid an incompatibility is polluting the name space of packages, and is not helpful.

Pascal summarizes the discussion that we are not going to avoid trying to fix problems in packages. Bigger changes might be a problem, but that is a case-by-case issue. John wonders if part of this meta-discussion should be at WG 9 level. Pascal thinks it would be better to push AIs forward with particular issues.

So assign this. Randy will take this.

Randy wonders if we should try to handle case sensitivity issues. Bibb suggests file name compares. Randy suggests that for creating names, you need to know not to use names that differ by just case.

Tucker suggests that maximum file name lengths and portions also be defined (as constants? As functions? As magic incantations??).