Final Minutes of the 10th ARG Meeting

29-31 March 2000
Phoenix AZ
USA

Attendees: John Barnes, Randy Brukardt, Gary Dismukes, Kiyoshi Ishihata, Mike Kamrad, Stephen Michell, Erhard Ploedereder, Joyce Tokar.

Meeting Summary

The meeting convened on 29 March 00 at 09:00 hours at DDC-I offices in Phoenix AZ and adjourned at 16:00 hours on 31 March.

With the exception of the last hour or so of the last day, all the time of the meeting was devoted to the review of the Corrigendum. All the wording changes of the Annexes were reviewed and the wording for them were changed as appropriate. The second draft of wording changes for the Core was reviewed and revised as appropriate. Review of the Record of Response and the Defect Reports is postponed until the June Meeting at Potsdam. Erhard will assign pages from both to members to review.

The remaining hour or so of the last day was devoted to reviewing several AIs.

The ARG unanimously thanked Joyce Tokar and DDC-I for the generous hospitality in hosting the meeting.

AI Summary

In the allotted time on the last day, these AIs were discussed and approved with minor wording changes:

AI-221 – Default_Bit_Order is static
AI-223 – terminators are skipped by Get

This amendment AI was discussed and it will require further rewrite and review:

AI-224 – Unsuppress

Next Meeting

The next meeting will be held in Potsdam, Germany, at the hotel of the Ada-Europe Conference on the afternoon of 30 June 2000 and all day on 1-2 July 2000 to complete the review of the documents. Erhard will make the local arrangements. There is a possibility for a Sept/Oct meeting to handle any remaining comments from WG9.

Action Items

There was no review of old action items, as the Technical Corrigendum took precedence. Few had been done anyway. For the record, all actions items are repeated here, even if already done. A review will follow at one of the next meetings.

Old action items on AIs remain:

Randy Brukardt: new AIs on Unsuppress and on `Object_size
Norm Cohen: AI-133, AI-167, AI-173
Robert Dewar: AI-85 (distribute an ‚append" Test to comp.lang.ada and collate results to AI)

Gary Dismukes: AI-158, AI-171 (rewrite wording), AI-196
Robert Eachus: AI-100, AI-172, AI-174, AI-185, AI-186
Mike Kamrad: new AI on Assert pragma
Pascal Leroy: AI-186(?), Streams, AI-108, AI-137/DR 40 + AI-117/DR 10 (partially)
Stephen Michell: AI-148, AI-187
Erhard Ploedereder: AI-171 (rewrite wording), AI-209, AI-212
Tucker Taft: AI-162, AI-188, AI-189, AI-191, AI-195(2.part), AI-216

All: Create tests for assigned AIs
Bob Duff: Be the test creator of last resort, distribute Design Note on „inward closures“
Randy Brukardt: distribute ‚inward-closures" study from prototyping project

New action items to be completed before the next meeting are:

Randy Brukardt: Update the next draft of the Corrigendum with the many detailed changes recorded at the meeting and in these minutes;
All: Review assigned pages of the Record of Response and Defect Report
Tucker Taft, Pascal Leroy: check out the new 3.7.1(7) AI-168
Pascal Leroy, Erhard Ploedereder, Pascal Leroy: check out consequences of DR09/AI-137
Erhard Ploedereder, Gary Dismukes: check AI-117 wording
Erhard Ploedereder: Publish logistics for the next meeting

Detailed Review of the Individual Corrigendum Items

This section describes the changes that were recommended to the new wording of each item listed in the core language Corrigendum. Only those paragraph whose new wording was changed are listed here; consider all others as being unchanged or so seriously affected by the recorded observations that a change at the meeting was not feasible. Each of the changes are listed by their paragraph number and Defect Report number and AI. Where the changes are minor, they are noted by underlined, emboldened font for easy identification.

The motivation for many of the changes was to be more concise and precise and this is the implied reason for the listed changes when no explicit reason is given.

3.6 (22 V1)/0002/AI-171

Erhard and Gary will review their private correspondence to determine whether the change in this replacement paragraph is correct. Otherwise there was a minor change:

„The elaboration of a discrete_subtype_definition that does not contain any per-object expressions creates the discrete subtype, and consists of the elaboration of the subtype_indication or the evaluation of the range. The elaboration of a discrete_subtype_definition that **<u>contains</u>** one or more per-object expressions is defined in 3.8. The elaboration of a component_definition in an array_type_definition consists of the elaboration of the subtype_indication. The elaboration of any discrete_subtype_definitions and the elaboration of the component_definition are performed in an arbitrary order.“

3.7.1 (7 V1)/0008/AI-168

Pascal wrote the additional sentence differently from the sentence in the minutes but he left no notes on why. Lots of time was spent determining how the two sentences differ. The example that Bob Eachus provided attempted to show an additional problem beyond the AI example and it was re-analyzed to find out what about it might have caused Pascal to change the wording. Here is the example:

```
package P is
   type T is private;
private
```

```
   type T (D: Integer := …) is …;
end P;

package P.Child is
   type NT is new P.T;         -- partial view is constrained
   type ANT is access all NT;  -- ditto
private
   X : NT (1);  -- legal and full view of NT is unconstrained
   subtype SANT is ANT (1);  --  illegal by corrigendum
end P.Child;
```

As to proposed wording, it is clear that only one place is necessary to cause this illegality and not many. Also the term „appears“ in the last sentence is too loosely defined. After more discussion, the illegality occurs when the view of the designated subtype is constrained. The recommended wording is:

> „A discriminant_constraint is only allowed in a subtype_indication whose subtype_mark denotes either an unconstrained discriminated subtype, or an unconstrained access subtype whose designated subtype is an unconstrained discriminated subtype. However, in the case of a general access subtype, a **discriminant_constraint** is illegal if there **is a place** within the immediate scope of the designated subtype where **its view is constrained**.“

3.8 (18 & 18.1)/0002/AI-171
Some wording changes were made on the inserted paragraph 18.1 to be more concise:

> „When a per-object constraint is elaborated (as part of creating an object), **each** per-object expression **of the constraint is** evaluated. For other expressions, **the** values determined during the elaboration of the component_definition or entry_declaration are used. Any checks associated with the enclosing subtype_indication or discrete_subtype_definition are performed, including the subtype compatibility check (see 3.2.2), and the associated subtype is created.“

Final approval on wording of both paragraphs was tabled until Gary and Erhard have had a chance to review them along with their private conversations.

4.5.2 (32.1 V1)/0016/AI-123
Remove the hyphen in the word, „non-limited“.

6.3.1 (2 V1)/0011/AI-117
Add the sentence previously proposed for inclusion in B.1(28.1 V1) /AI-117 here, because its content is more relevant to the Core than the Annex.

> „As explained in B.1, ``Interfacing Pragmas", a *convention* can be specified for an entity. **Unless this International Standard states otherwise, the default convention of an entity is Ada.** For a callable entity or access-to-subprogram type, the convention is called the *calling convention*. The following calling conventions are defined by the language:“

7.6.1 (13 V1)/0021 & 0022/AI-169 & AI-193
The replacement paragraph did not include any of the changes from the last meeting. With those changes the paragraph now reads:

> „If the object_name in an object_renaming_declaration, or the actual parameter for a generic formal **in out** parameter in a generic_instantiation, denotes any part of an anonymous object created by a function call, the anonymous object is not finalized until after it is no longer accessible via any name. Otherwise, an anonymous object created by a function call or by an aggregate is finalized no later than the end of the innermost enclosing declarative_item or statement; if that is a compound_statement, it is finalized before starting the execution of any statement within the compound_statement. If a transfer of control or

raising of an exception occurs prior to performing a finalization of an anonymous object, the anonymous object is finalized as part of the finalizations due to be performed for the object's innermost enclosing master.“

7.6.1 (20.1 V1)/0024/AI-83
Minor changes to improve readability:

„*Implementation Requirements*

For an aggregate of a controlled type whose value is assigned, other **than by** an assignment_statement or **a** return_statement, the implementation shall not create a separate anonymous object for the aggregate. The aggregate value shall be constructed directly in the target of the assignment operation and Adjust is not called on the target object.“

The merits of AI-83 in its current form were briefly discussed again, because of general concern about omitted implicit calls on Initialize, Adjust, and Finalize, and because of the inclusion of the minor correction to AI-83 that AI-197 (unseen by WG9) introduced. However, no alternative proposal were made, so the changes stay as decided.

8.5.1 (5 V1)/0017/AI-184
The latest draft put the word „derived“ back into the phrase „generic formal [derived] type“ at the end of the paragraph. After some discussion, it was concluded that the minutes were incorrect and that the latest draft is correct. With minor changes, the paragraph now reads:

„The renamed entity shall not be a subcomponent that depends on discriminants of a variable whose nominal subtype is unconstrained, unless this subtype is indefinite, or the variable is aliased. A slice of an array shall not be renamed if this restriction disallows renaming of the array. 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. These rules also apply for a renaming that appears in the body of a generic unit, with the additional requirement that even if the nominal subtype of the variable is indefinite, its type shall not be a descendant of **an** untagged generic formal derived type.“

A "but" was dropped in the 6th line (it read "..unit, but with...").

8.5.4 (5 V1)/0027 & 0028/AI-135
A small editorial improvement was made:

"...the renamed subprogram**. O**therwis**e,** the profile...."

For more substantive problems, see the later discussion on DR27/AI-135.

8.5.4 (8, 8.1, 8.2 V1)/0014 & 0027/AI-64 & AI-135
A discrepancy between the minutes and the revised text was noted. In this case, the minutes were correct. The fix is:

„For a call to a subprogram whose body is given as a renaming-as-body, the execution of the renaming-as-body is equivalent to the execution of a subprogram_body that simply calls the renamed subprogram with its formal parameters as the actual parameters and, if **it is** a function, returns the value of the call.

For a call on a renaming of a dispatching subprogram that is overridden, if the overriding occurred before the renaming, then the body executed is that of the overriding declaration, even if the overriding declaration is not visible at the place of the renaming; otherwise, the inherited or predefined subprogram is called.

Bounded (Run-Time) Errors

If a subprogram directly or indirectly renames itself, then it is a bounded error to call that subprogram. Possible consequences are that Program_Error or Storage_Error is raised, or that the call results in **infinite recursion**."

9.10 (6.1 V1)/0031/AI-118
Using the precedent defined by 9.10 (15), the wording is changed accordingly:

„• If A1 is the termination of a task T, and A2 is **either** the evaluation of the expression T'Terminated or a call to Ada.Task_Identification.Is_Terminated **with an actual** parameter **that** identifies T **(see C.7.1)**;"

10.1.4 (4 V1)/0032/AI-192
„If a library_unit_body that is a subprogram_body is submitted to the compiler, it is interpreted only as a completion if a library_unit_declaration with the same defining**_program_**unit_name already exists in the environment for a subprogram other than an instance of a generic subprogram or for a generic subprogram (even if the profile of the body is not type conformant with that of the declaration); otherwise the subprogram_body is interpreted as both the declaration and body of a library subprogram."

10.1.5 (7.1 V1)/0034 & 0035/AI-41 & AI-2
The discussion yielded agreement on the new wording which was provided by Erhard shortly after the meeting:

„*Static Semantics*
A library unit pragma that applies to a generic unit does not apply to its instances, unless a specific rule for the pragma specifies the contrary.

Implementation Advice
When applied to a generic unit, a program unit pragma that is not a library unit pragma should apply to each instance of the generic unit for which there is not an overriding pragma applied directly to the instance."

12.5 (8 V1)/0038/AI-202
Replace „operations" with „operators", making the paragraph now read:

„The formal type also belongs to each class that contains the determined class. The primitive subprograms of the type are as for any type in the determined class. For a formal type other than a formal derived type, these are the predefined operators of the type. For an elementary formal type, the predefined **operators** are implicitly declared immediately after the declaration of the formal type. For a composite formal type, the predefined **operators** are implicitly declared either immediately after the declaration of the formal type, or later in its immediate scope according to the rules of 7.3.1. In an instance, the copy of such an implicit declaration declares a view of the predefined operator of the actual type, even if this operator has been overridden for the actual type. The rules specific to formal derived types are given in 12.5.1."

13.1 (8.1 V1)/0009/AI-137
The wording change was made to coincide with the style of 13.1 (8):

„**An operational item *directly specifies* an *operational aspect* of the type of the subtype denoted by the local_name.** The local_name of an operational item shall denote a first subtype. Operational items are type-related."

13.3 (9 V1)/0009/AI-137
„The following representation attributes are defined**: Address, Alignment, Size, Storage_Size, and Component_Size**."

13.3 (74 & 74.1 V1)/0009/AI-137
„The following operational attribute is defined: **External_Tag**.

For every subtype S of a tagged type T (specific or class-wide):“

13.11 (2 V1)/0009/AI-137
This replacement paragraph was dropped from the Corrigendum because the text is non-normative.

13.12 (9.1 V1)/0043 & 0044/AI-190 & AI-181
„**An** implementation **is** permitted to omit restriction checks for code that is recognized at compile time to be unreachable and for which no code is generated.“

13.13.2 (1 V1)/0009/AI-137
„The **operational attributes** Write, Read, Output, and Input convert values to a stream of elements and reconstruct values from a stream.“

13.14 (8.2 V2)/0047/AI-81
„An implicit call freezes the same entities that would be frozen by an explicit call. This is true even if the implicit call is removed via **i**mplementation **p**ermissions.“

A (2 V1)/0048/AI-81
Replace the reference to A.10.8 with A.10.9 and the reference to A.10.9 with A.10.8.

A.4.3 (68 V1)/0050/AI-128
Replace the last sentence of the paragraph with:

„The value returned for First is Source’First, provided Source’First is in Positive, otherwise a Constraint_Error is raised.“

Additionally, add a missing right parenthesis to the third item of the question in the associated Defect Report.

A.4.3 (74 V1)/0050/A-128
There are too many uses of the word „otherwise“ which is confusing to read; eliminate the last one. And then put the last two if statements into a bulleted list, making the replacement paragraph read:

„If Low > Source'Last+1, or High < Source'First-1, then Index_ Error is propagated. Otherwise:

- If High >= Low, then the returned string comprises Source(Source'First..Low-1) & By & Source(High+1..Source'Last), but with lower bound 1.
- If High < Low, then the returned string is Insert(Source, Before=>Low, New_Item=>By).“

A.4.3 (86 V1)/0050/A-128
Drop the word „but“.

A.4.3 (106 V1)/0050/A-128
The last sentence of the replacement paragraph should read:

„The second function returns a string whose length is Left*Right'Length and whose value is the null string if Left = 0 and **otherwise** is (Left-1)*Right & Right with lower bound.“

A.4.4 (105 V1)/0050/A-128
„Each of the transformation subprograms (Replace_Slice, Insert, Overwrite, Delete), selector subprograms (Trim, Head, Tail), and constructor functions ("*") has an effect based on its corresponding subprogram in Strings.Fixed, and Replicate is based on Fixed."*". **In the case of a function**, the corresponding fixed-length string function is applied to the string represented by the Bounded_String parameter. To_Bounded_String is applied to the result string, with Drop (or Error in the case of Generic_Bounded_Length."*") determining the effect when the string length exceeds Max_Length. **In the**

case of a procedure, the corresponding function in Strings.Bounded.Generic_Bounded_Length is applied, with the result assigned into the Source parameter.“

A.5.1 (9 V1)/0020/AI-126
Packages don't define pragmas they use pragmas; consequently change the wording to say the package is declared pure to mimic pragma of the generic definition. The replacement paragraph now reads:

„The library package Numerics.Elementary_Functions **is declared pure and defines the same** subprograms as Numerics.Generic_Elementary_Functions, except that the predefined type Float is systematically substituted for Float_Type'Base throughout. Nongeneric equivalents of Numerics.Generic_Elementary_Functions for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Elementary_Functions, Numerics.Long_Elementary_Functions, etc.“

A.5.2 (40 V1)/0051/AI-89
A heading, "Bounded (Run-Time) Errors", is added ahead of the new paragraph and some minor corrections are made to the paragraph. It now reads:

„Bounded (Run-Time) Errors

It is a bounded error to invoke Value with a string that is not the image of any generator **state**. If the error is detected, Constraint_Error or Program_Error is raised. Otherwise, a call to Reset with the resulting state will produce a **generator** such that calls to Random with this **generator** will produce a sequence of values of the appropriate subtype, but which might not be random in character. That is, the sequence of values might not fulfill the implementation requirements of this clause.“

A.10.3 (22 & 23 V1)/0054/AI-63
It appears that much of paragraph 23 duplicates the substance of 22 except for the fact that the file is closed as opposed to no longer existing. So why not add the file closing as an alternative to 22 to make it clearer. The "no longer exists" part of the sentence handles the cases of leaving the scope of the respective file object or of its unchecked deallocation; the problem of external files being screwed by sharing in the environment is handled elsewhere in the RM. Therefore it appears OK to combine the two paragraphs. Consequently, the Corrigendum will have a new replacement paragraph 22 as follows:

„The execution of a program is erroneous if it invokes an operation on a current default input, default output, or default error file if the corresponding file object is closed or no longer exists.“

And the Corrigendum will show that paragraph 23 is deleted.

A.12.1 (28-36 V1)/0056&0057/AI-26&AI-1
Initially the discussion focused on the individual replacement paragraphs 28, 29, 30, 32 and 35. The wordsmithing oscillated among these paragraphs, not successfully resolving the impact of indexing.

This section introduces the orthogonal concept of meaning of index for positional files, in particular the existence of current index. It became clear that the conceptual model in this section is not understood and therefore a chalkboard model was created as follows:

Capability	Finite positional stream	Finite non-positional stream
Start	Yes	Yes
End	Yes	Yes
Specific position	Yes	No – user_error
Size	Yes	Implementation dependent
Index & Set_Index	Yes	No – user_error
Append	Yes	Yes

Reset w/o Append Mode	Like Sequential and index = 1	Like Sequential
Reset w/ Append Mode	Size (file) + 1	After end of file

Randy was asked to rewrite this whole section (overnight) with the goal of reusing as much of the section on sequential IO and of adding new material (per table above) on the effects of indexing. To set up the formal model, it was decided that the first paragraph of the Static Semantics subsection should explain the effects of positioning, per this table.

Further tuning was done to the draft paragraphs that Randy presented. To avoid unnecessarily repeating the first four paragraphs of A.8, the two paragraphs following the package specification draw implications through the reference to sections on Sequential_IO. The definition of direct access file and sequential access file are unnecessary because the reference should be sufficient. Therefore the effects of positioning can be directly tied to the existence of current index in the text.

It was also decided that, for presentation purposes, all the descriptions of the operations of this package should be separated from and precede the description of the impact that positioning has on the current index.

The resulting changes to the Corrigendum are as follows:

„Insert before paragraph 2: [8652/0055]

The library package Streams.Stream_IO has the following declaration:

the new paragraph:

The elements of a stream file are stream elements. If positioning is supported for the specified external file, a current index and current size are maintained for the file as described in A.8. If positioning is not supported, a current index is not maintained and the current size is implementation-defined.

Insert after paragraph 28: [8652/0055] *{which is unchanged from original RM}*

The subprograms Create, Open, Close, Delete, Reset, Mode, Name, Form, Is_Open, and End_of_File have the same effect as the corresponding subprograms in Sequential_IO (see A.8.2).

the new paragraphs:

The Set_Mode procedure changes the mode of the file. If the new mode is Append_File, the file is positioned to its end; otherwise, the position in the file is unchanged. *(formerly the original RM paragraph 35)*
The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File. *{formerly the original RM paragraph 36}*

Replace paragraph 29: [8652/0056]

The Stream function returns a Stream_Access result from a File_Type object, thus allowing the stream-oriented attributes Read, Write, Input, and Output to be used on the same file for multiple types.

by:

The Stream function returns a Stream_Access result from a File_Type object, thus allowing the stream-oriented attributes Read, Write, Input, and Output to be used on the same file for multiple types. Stream propagates Status_Error if File is not open.

Insert after paragraph 30: [8652/0055] *{which is unchanged from original RM}*

The procedures Read and Write are equivalent to the corresponding operations in the package Streams. Read propagates Mode_Error if the mode of File is not In_File. Write propagates Mode_Error if the mode of File is not Out_File or Append_File. The Read procedure with a Positive_Count parameter starts reading at the specified index. The Write procedure with a Positive_ Count parameter starts writing at the specified index.

the new paragraph:

The Size function returns the current size of the file.

Replace paragraph 31: [8652/0055]

The Index function returns the current file index, as a count (in stream elements) from the beginning of the file. The position of the first element in the file is 1.

by:

The Index function returns the current index.

Insert after paragraph 32: [8652/0055] *{which is unchanged from original RM}*

The Set_Index procedure sets the current index to the specified value.

the new paragraphs:

If positioning is supported for the external file, the current index is maintained as follows:

- For Open and Create, if the Mode parameter is Append_File, the current index is set to the current size of the file; otherwise, the current index is set to one.
- For Reset, if the Mode parameter is Append_File, or no Mode parameter is given and the current mode is Append_File, the current index is set to the current size of the file plus one; otherwise, the current index is set to one.
- For Set_Mode, if the new mode is Append_File, the current index is set to current size plus one; otherwise, the current index is unchanged.
- For Read and Write without a Positive_Count parameter, the current index is incremented by the number of stream elements read or written.
- For Read and Write with a Positive_Count parameter, the value of the current index is set to the value of the Positive_Count parameter plus the number of stream elements read or written.

{Paragraph 33 remains unchanged from the original RM}

Delete paragraph 34: [8652/0055]

The Size function returns the current size of the file.

Delete paragraph 35: [8652/0055]

The Set_Mode procedure changes the mode of the file. If the new mode is Append_File, the file is positioned to its end; otherwise, the position in the file is unchanged.

Replace paragraph 36: [8652/0056; 8652/0055]

The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File.

by:

Erroneous execution

If the File_Type object passed to the Stream function is later closed or finalized, and the stream-oriented attributes are subsequently called (explicitly or implicitly) on the Stream_Access value returned by Stream, execution is erroneous. This rule applies even if the File_Type object was opened again after it was closed.“

Note that the existence of infinite stream files to cover communication such as http became apparent. This represents a whole new capability outside the scope of this section and should be the topic of an amendment AI.

A.14 (3)/0058/AI-50

The deletion of this paragraph is approved but it highlighted the impact on the RM by deleted paragraphs. It was decided that the most expedient and revealing approach is to indicate with special notation alongside the paragraph number that the paragraph is deleted.

B.1 (9.1 V1)/0059/AI-36
Remove junk punctuations, so that the new inserted paragraph reads:

> „For pragmas Import and Export, the argument for Link_Name shall not be given without the pragma_argument_identifier unless the argument for External_Name is given.“

B.1 (28 V1)/0011/AI-117
Move last sentence to become the second sentence of paragraph 6.3.1 (2) because its content should not be hidden in an appendix. Consequently, there is no replacement paragraph in the Corrigendum.

B.3(60 V1)/0060/AI-131
The last sentence of the second inserted paragraph is really normative information and should then appear at the beginning of the new paragraph. Along with some additional typo fixes, the first two inserted paragraphs should read:

> „A Convention pragma with *convention*_identifier C_Pass_By_Copy **shall** only be applied to a type.
>
> **The eligibility rules in B.1 do not apply to convention C_Pass_By_Copy. Instead,** a type T is eligible for convention C_Pass_By_Copy if T is a record type that has no discriminants and that only has components with statically constrained subtypes, and each component is C-compatible.“

B.3 (63.1 V1)/0061/AI-37
This new inserted paragraph is OK but, as a note to Randy, make sure that the introduction of the TC defines the positioning convention when the paragraph is inserted before a paragraph that has a heading.

B.3 (68.1 V1)/0060/AI-131
The new inserted paragraph is:

> „• An Ada parameter of a C_Pass_By_Copy-compatible (record) type T, of mode **in**, **is** passed as a t argument to a C function, where t is the C struct corresponding to the Ada type T.“

B.3.1 (24 V1)/0062/AI-140
Minor wordsmithing:

> „If Item is **null**, then To_Chars_Ptr returns Null_Ptr. **If Item is not null,** Nul_Check is True, and Item.**all** does not contain nul, then the function propagates Terminator_Error; otherwise To_Chars_Ptr performs a pointer conversion **without** allocation of memory.“

B.3.1 (36 V1)/0063/AI-139
Minor wordsmithing:

> „If Item = Null_Ptr**,** then Value propagates Dereference_Error. Otherwise**,** Value returns the shorter of two arrays, **either** the first Length chars pointed to by Item, **or** Value(Item). The lower bound of the result is 0. If Length is 0**,** then Value propagates Constraint_Error.“

The discussion brings out the issue of additional wordsmithing for the paragraph under examination and similar paragraphs, beyond the changes being applied. The recommendation is to do the wordsmithing when it makes the understanding better. The writer of the new RM is given the discretion to make the changes to associated paragraphs.

B.4 (71&79 V1)/0068/AI-72
In both paragraphs, the reference to „Num“ should be to the parameter identifier, „Item“, so that paragraph 71 should read:

„This function returns the Numeric value for Item, represented in accordance with Format. The length of the returned value is Length(Format), and the lower bound is 1. Conversion_Error is propagated if **Item** is negative and Format is Unsigned.“

And paragraph 78 should read:

„This function returns the Packed_Decimal value for Item, represented in accordance with Format. The length of the returned value is Length(Format), and the lower bound is 1. Conversion_Error is propagated if **Item** is negative and Format is Packed_Unsigned.“

C.3.1 (12 V1)/0069/AI-121
The word „Otherwise“ is dropped from the last sentence as the only other remaining condition is covered by the new paragraph, C.3.1 (14.1) below; and there are other minor changes. The replacement paragraph now reads:

„When a protected object is finalized, for any of its procedures that are attached to interrupts, the handler is detached. If the handler was attached by a procedure in the Interrupts package or if no user handler was previously attached to the interrupt, the default treatment is restored. If an Attach_Handler pragma was used and the most recently attached handler for the same interrupt is the same as the one that was attached at the time the protected object was **initialized**, the previous handler is restored.“

C.3.1 (14.1 V1)/0069/AI-121
Minor changes to improve readability:

„If the handlers for a given interrupt attached via pragma Attach_Handler are not attached and detached in a stack-like (LIFO) order, program execution is erroneous. In particular, when a protected object is finalized, **the execution is erroneous** if any of **the** procedures **of the protected object** are attached to interrupts via pragma Attach_Handler **and** the most recently attached handler for the same interrupt is not the same as the one that was attached at the time the protected object was **initialized**.“

C.3.2 (16 V1)/0070/AI-166
The second half of the last sentence is dropped as it is redundant with C.3.2 (17); it now reads:

„The Current_Handler function returns a value that represents the attached handler of the interrupt. If no user-defined handler is attached to the interrupt, Current_Handler returns **null**.“

C.3.2 (18 V1)/0070/AI-166
The word „null“ in the last sentence is emboldened.

C.7.2 (13.1 V1)/0072/AI-165
Remove the word „it“ from the last sentence of the new paragraph; it now reads:

„*Bounded (Run-Time) Errors*

If the package Ada.Task_Attributes is instantiated with a controlled type and the controlled type has user-defined Adjust or Finalize operations that in turn access task attributes by any of the above operations, then a call of Set_Value of the instantiated package constitutes a bounded error. The call may perform as expected or may result in forever blocking the calling task and subsequently some or all tasks of the partition.“

C.7.2 (15.1 V1)/0072/AI-165

„Accesses to task attributes via a value of type Attribute_Handle are erroneous if executed concurrently with each other or with calls of any of the **operations declared in package Task_Attributes**.“

C.7.2 (16 V1)/0072/AI-165
New wording was added to be more specific:

„**For a given attribute of a given task,** the implementation shall perform the operations **declared in this package** atomically with respect to any of these operations of the same attribute of the same task. The granularity of any locking mechanism necessary to achieve such atomicity is implementation defined.“

D.7 (17 V1)/0077/AI-67
New wording was added to be more specific:

„Max_Storage_At_Blocking
Specifies the maximum portion (in storage elements) of a task's Storage_Size that can be retained by a blocked task. If an implementation chooses to detect a violation **of this restriction**, Storage_Error should be raised; otherwise, the behavior is implementation defined.“

D.7 (18 V1)/0077/AI-67
New wording was added to be more specific:

„Max_Asynchronous_Select_Nesting
Specifies the maximum dynamic nesting of asynchronous_selects. A value of zero prevents the use of any asynchronous_select and, if a program contains an asynchronous_select, it is illegal. If an implementation chooses to detect a violation **of this restriction** for values other than zero, Storage_Error should be raised; otherwise, the behavior is implementation defined.“

D.7 (19 V1)/0077/AI-67
New wording was added to be more specific:

„Max_Tasks
Specifies the maximum number of task creations that may be executed over the lifetime of a partition, not counting the creation of the environment task. A value of zero prevents any task creation and, if a program contains a task creation, it is illegal. If an implementation chooses to detect a violation **of this restriction** for values other than zero, Storage_Error should be raised; otherwise, the behavior is implementation defined.“

E.2.2 (9 V1)/0082 & 0083/AI-4 & AI-164
A bulleted list was added to the paragraph to be more specific and to improve readability:

„An access type declared in the visible part of a remote types or remote call interface library unit is called a *remote access type*. Such a type shall be:

- An access-to-subprogram type**, or**
- A general access type that designates a class-wide limited private type or a class-wide private type extension **all of** whose ancestors are **either** private type extensions **or** limited private type**s**.

A type that is derived from a remote access type is also a remote access type.“

E.2.2 (14 V1)/0084/AI-47
Remove the hyphen from word „non-limited“.

E.2.2 (17)
For the updated Reference Manual, remember to remove the unnecessary semicolon at the end of the paragraph.

E.2.3 (7 V1)/0079/AI-48
New wording was added to be more specific:

„A *remote call interface (RCI)* is a library unit to which the pragma Remote_Call_Interface applies. A subprogram declared in the visible part of such a library unit, or **declared** by such a library unit, is called a *remote subprogram*.“

The confusion about the phrase, „declared by such a library unit“ (which is the new wording of the replacement paragraph) touched off a discussion about why the RCI pragma on a generic is not inherited by its instantiations and whether an instantiation could be RCI without the generic being RCI and whether this could not lead to an unintentional hole in the semantics. (No example was presented, though.) Randy will start an AI on this.

E.2.3 (9 V1)/0079/AI-48

> „In addition, the following restrictions apply to **an** RCI library unit:“

E.4 (20.1 V1)/0087/AI-82

New wording was added to be more specific:

> „With respect to shared variables in shared passive library units, the **execution of the** corresponding subprogram body of **a** synchronous remote procedure call is considered to be part of the execution of the calling task. The execution of the corresponding subprogram body of an asynchronous remote procedure call proceeds in parallel with the calling task **and** does not signal the next action of the calling task (see 9.10).“

E.5 (24.1 V1)/0088/AI-82

In the first new paragraph, the sentences were rearranged to emphasize the point that the possibility of changes in the implementation permissions of the introduction to Annex A will not unnecessarily affect this paragraph. And some additional wording changes were made to be more specific:

> „An implementation shall not restrict the replacement of the body of System.RPC. An implementation shall not restrict children of System.RPC. The **related implementation** permissions **in** the introduction to Annex A **do** not apply.“

The second new paragraph was restructured as a conditional statement to emphasize importance of supporting remote subprogram calls using a user-defined System.RPC as defined in this Annex:

> „**If** the implementation of System.RPC **is** provided by the user, an implementation shall support remote subprogram calls **as specified**.“

G.1.1 (25 V1)/0020/AI-126

Packages don’t define pragmas; they use pragmas. Consequently change the wording to say the package is declared pure to mimic pragma of the generic definition. The replacement paragraph now reads:

> „The library package Numerics.Complex_Types **is declared pure and defines** the same types, constants, and subprograms as Numerics.Generic_Complex_Types, except that the predefined type Float is systematically substituted for Real'Base throughout. Nongeneric equivalents of Numerics.Generic_Complex_Types for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Types, Numerics.Long_Complex_Types, etc.“

G.1.2 (9 V1)/0020/AI-126

Packages don’t define pragmas; they use pragmas. Consequently change the wording to say the package is declared pure to mimic pragma of the generic definition. The replacement paragraph now reads:

> „The library package Numerics.Complex_Elementary_Functions **is declared pure and defines** the same subprograms as Numerics.Generic_Complex_Elementary_Functions, except that the predefined type Float is systematically substituted for Real'Base, and the Complex and Imaginary types exported by Numerics.Complex_Types are systematically substituted for Complex and Imaginary, throughout. Nongeneric equivalents of Numerics.Generic_Complex_Elementary_Functions corresponding to each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Elementary_Functions, Numerics.Long_Complex_Elementary_Functions, etc.“

G.1.3 (12 V1)/0093/AI-29
It was felt that introducing the new term, „real numbers“, was unnecessary; therefore the second sentence was modified to be more specific about the relationship of the format to the corresponding Get procedure of Text_IO.Float_IO for the base subtype of Complex_Types.Real. Another change was to remove the „and/or“ from the third sentence:

> „The input sequence is a pair of optionally signed real values representing the real and imaginary components of a complex value. These components **have the format defined for the corresponding Get procedure of** an instance of Text_IO.Float_IO (see A.10.9) **for the base subtype of Complex_Types.Real**. The pair of components may be separated by a comma **or** surrounded by a pair of parentheses **or both**. Blanks are freely allowed before each of the components and before the parentheses and comma, if either is used. If the value of the parameter Width is zero, then“

J.7.1 (16 V1)/0078/AI-111

> „Interrupt entry calls may be implemented by having the hardware **directly execute** the appropriate accept_statement. Alternatively, the implementation is allowed to provide an internal interrupt handler to simulate the effect of a normal task calling the entry.“

Detailed Review of the Individual Defect Reports

There were a handful of Defect Reports that deserved focused review because of their distributed impact throughout the Reference Manual.

DR9/AI-137
At the center of the proposal made by Pascal is the introduction of a new syntax production, aspect_clause, to tie together representation_clause and attribute_definition_clause. This change will be a significant disruption to existing compilers and tools; in particular, the ASIS standard and tools will be affected by the proposed change. Many object to such a large change; making such structural syntax changes should be reserved for significant additions to the language and not cleaning up problems in the existing language.

Erhard would prefer to limit the change to the renaming of the nonterminal, representation_clause, to aspect_clause in the existing production for representation_clause. The change in the text and the substitutions in all syntax productions where representation_clause is referenced would remain. This eliminates the offending production and the significant disruption to existing compilers and tools. The counter proposal would just increase the number of kinds of representation items to include all six kinds, namely, attribute_definition_clauses for representation attributes, enumeration_representation_clauses, record_representation_clauses, at_clauses, component_clauses, and *representation pragmas*.

It was noted that two paragraphs needed to be added to the list of paragraphs covered by this Defect Report, namely, 9.1(12) and 13.4(11).

Erhard had the further worry that the "narrowing" of the meaning of representation item might have unintentional semantic effects elsewhere in the RM (e.g., a rule on representation items that should apply to all attribute definition clauses, but no longer does, because operational attribute definition clauses are now excluded.). Also, for the change to aspect_clause, a global check on the RM is needed to make sure that all instances are caught.

In summary, here is the list of paragraphs from the Defect Report that are revised by this discussion:

13.1 Representation Items

Replace the title: [8652/0009]

Representation Items

by:

Operational and Representation Items

Replace paragraph 1: [8652/0009]

> There are three kinds of *representation items*: representation_clauses, component_clauses, and *representation pragmas*. Representation items specify how the types and other entities of the language are to be mapped onto the underlying machine. They can be provided to give more efficient representation or to interface with features that are outside the domain of the language (for example, peripheral hardware). Representation items also specify other specifiable properties of entities. A representation item applies to an entity identified by a local_name, which denotes an entity declared local to the current declarative region, or a library unit declared immediately preceding a representation pragma in a compilation.

by:

> There are **six** kinds of *representation items*: attribute_definition_clauses for representation attributes, **enumeration_representation_clauses, record_representation_clauses, at_clauses**, component_clauses, and *representation pragmas*. Representation items specify how the types and other entities of the language are to be mapped onto the underlying machine. They can be provided to give more efficient representation or to interface with features that are outside the domain of the language (for example, peripheral hardware). An attribute_definition_clause for an operational attribute is an *operational item*. Operational items specify other specifiable properties of entities. An operational item or a representation item applies to an entity identified by a local_name, which denotes an entity declared local to the current declarative region, or a library unit declared immediately preceding a representation pragma in a compilation.

Replace paragraph 2: [8652/0009]

```
representation_clause ::= attribute_definition_clause
     | enumeration_representation_clause
     | record_representation_clause
     | at_clause
```

by:

```
aspect_clause ::= attribute_definition_clause
     | enumeration_representation_clause
     | record_representation_clause
     | at_clause
```

Replace paragraph 4: [8652/0009]

> A representation pragma is allowed only at places where a representation_clause or compilation_unit is allowed.

by:

> A representation pragma is allowed only at places where an **aspect_clause** or compilation_unit is allowed.

DR14/AI-64

There is an apparent freezing problem that the renaming-as-body creates in the new 3.11.1(1 V1) because of the presence of „noninstance body“ in 13.14(3) causes unintended consequences on renaming-as-body. See DR27/AI-135. The proposed solution is to modify AI-64/DR14 to include the following new wording for 13.14(3):

> „A noninstance body **other than a renaming-as-body** causes freezing of each entity declared before it within the same declarative_part.“

DR24/AI-83

Erhard complains that the last sentence of the summary in the Defect Report is misleading (besides having a typo):

> „No temporary object is create (sp.) and Adjust is not called on the object as a whole.“

"the object" binds badly to the temporary object. And, the usual terminology talks about an anonyomous object. The following replacement sentence was recommended:

„No anonymous objected is created and Adjust is not called on the target object.“

DR27/AI-135
There are three alternatives which were reviewed as the last sentence of 8.5.4 (5 V1):

Alternative 1: A renaming-as-body is illegal if the subprogram it declares names the subprogram itself or renames a subprogram that takes its convention from the subprogram it declares.

Alternative 2: It is illegal for a renaming-as-body that occurs before the subprogram it declares is frozen to rename itself either directly or indirectly through one or more renamings, none of which have been frozen.

Alternative 3: A renaming-as-body is illegal if the declaration occurs before the subprogram is frozen and the renaming renames the subprogram itself, either directly or indirectly.

Alternative 1 succinctly describes the symptom of the problem but a description of symptoms is improper as a legality rule and it has the problem of using the term „taking the convention“, which might be a mystery to users. Alternative 2 creates a recursive definition when it uses the phrase „to rename itself“ and also the phrases, „either directly or indirectly“ and „none of which have been frozen“, were considered too vague.
Alternative 3, which is the original wording for the inserted paragraph, has the vague phrase, „either directly or indirectly“. It also does not account for the possibility that one of the subprograms on the chain has been frozen and hence breaks the cyclic dependency in establishing a convention.

Alternative 1 quickly loses favor and is dropped, due to the original objections.

The phrase, „either directly or indirectly“ is dropped in Alternatives 2 and 3 in favor of the phrase, „through one or more renamings“. And then these modified Alternatives were tested against the following example:

```
package P is
   function F return T;
   O : T := F;  --  freezes F
End;

package body P is
   function G return T;
   function F return T renames G;  --  freezes G
end P;
```

It would appear that the modified Alternative 3 does the job with the replacement.

Then examination of 13.14(3) and the freezing rules showed a potential for solving the problem without any of the suggested Alternatives:

„The end of a declarative_part, protected_body, or a declaration of a library package or generic library package, causes freezing of each entity declared within it, except for incomplete types. A noninstance body causes freezing of each entity declared before it within the same declarative_part.“

The renaming-as-body is linked to this sentence through the Corrigendum change to paragraph 3.11.1(1 V1)/DR14/AI-64:

„Declarations sometimes come in two parts. A declaration that requires a second part is said to *require completion*. The second part is called the *completion* of the declaration (and of the entity declared), and is either another declaration, a body, or a pragma. A *body* is a body, an entry_body, or a renaming-as-body (see 8.5.4).“

Without further action, this means that a renaming_as_body (as a body) freezes everything above it and hence no cyclic dependency of convention inheritance is possible via multiple renaming_as_body declarations. This follows from the original wording in 8.5.4(5) and with the second sentence of the replacement paragraph.

> „If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the profile shall be mode-conformant with that of the renamed callable entity and the subprogram it declares takes its convention from the renamed subprogram; otherwise the profile shall be subtype-conformant with that of the renamed callable entity and the convention of the renamed subprogram shall not be Intrinsic.“

(Coincidentally, it was decided to split this sentence into two sentences at the „otherwise“.) This comes close to solving the dependency problem, but causes a significant restriction, essentially requring Convention pragmas immediately after the subprogram specification. That's hard to swallow. But if this is not the intent, then renaming_as_body better not be a body for the purposes of freezing -- a "non-instance body"? It certainly resembles one (akin to bodies of instantiations).

In an attempt to get around this problem, Erhard proposes new wording for 13.14(3):

> „A noninstance body **other than a renaming-as-body** causes freezing of each entity declared before it within the same declarative_part.“

Alternative 3 was then selected with the following new wording:

> „A renaming-as-body is illegal if the declaration occurs before the subprogram whose declaration **it completes** is frozen, and the renaming renames the subprogram itself, **through one or more subprogram renaming declarations, none of whose subprograms has been frozen**.“

DR-40/AI-108
This DR was discussed somewhat non-conclusively. As subsequent e-mail discussion has elaborated much better on the point and the issues, no report is given here.

Review of the Individual AIs

AI-221
There were some minor changes to fix spelling, in the syntax to remove the angle brackets surrounding implementation defined, to put that term in italics, and to remove all unnecessary hyphens. Approved 8-0-0.

AI-223
The solution that Randy proposed is based on wording from the Ada83. After some detailed discussion on its effect, it was decide that the phrase, „for this purpose“ should be removed from the replacement sentence. Approved 5-0-2.

Suppress/Unsuppress Amendment

Randy’s survey on the capability of inheriting suppress shows that Averstar and DDCI do provide it and Irvine, GreenHills, Rational, ACT and Janus do not.

Erhard doesn’t care to make a language change to specify the details for the optional, selective features of Suppress; implementors should figure out what they want to do here; the implementation permissions make the semantics completely ignorable anyway. But this would not the be the case for handling a pragma Unsuppress similarly allowing for selective unsuppressing, since this pragma makes sense only without a permission to ignore it. So, at best a parameterless Unsuppress might make sense for a standard (he thinks).

There was some discussion about the "inheritance" of Suppress/Unsuppress from package spec to body. No real conclusions were drawn. The sentiment that a non-selective Unsuppress for a local scope, i.e., a subprogram body, should work was quite apparent.