Minutes of 14th ARG Meeting

5-7 October 2001

Bloomington, Minnesota USA

Attendees: John Barnes, Randy Brukardt, Gary Dismukes, Robert Duff, Kiyoshi Ishihata, Mike Kamrad (all but Sunday morning), Pascal Leroy, Erhard Ploedereder, Tucker Taft.

Observers: Steve Baird (Rational, USA); Alan Burns (University of York, UK, Friday morning only)

Meeting Summary

The meeting convened on 5 October 2001 at 13:00 hours and adjourned at 16:00 hours on 7 October 2001. The meeting was held at the Thunderbird Hotel in a room provided by the SIGAda Conference.

This was the first meeting to be chaired by Pascal Leroy. It approved the ARG Procedures drafted by Randy and Pascal. The meeting covered the entire agenda. The first two days were largely spent on amendment AIs. The third day was largely spent on normal AIs.

By acclamation the meeting thanked the SIGAda Conference Committee for the facilities and the excellent supply of food and refreshments.

Next Meeting

There is a need to have another ARG meeting before the next Ada-Europe meeting in June 2002. Pascal recommended Rational in Cupertino, CA on 10-12 February 2002, which was accepted.

Old Action Items

The old action items were reviewed. Following is a list of items completed (other items remain open):

Steve Baird:

- AI-263

John Barnes:

- Downward closures

Randy Brukardt:

- AI-218
- AI-229
- AI-237
- AI-248
- AI-257
- AI-258
- AI-260
- AI-262
- AI-268

- Split second issue from AI-263 into a separate AI (new AI-269).

Gary Dismukes:

- AI-196

Pascal Leroy:

- AI-195
- AI-255

Tucker Taft:

- AI-216
- AI-230
- AI-231
- AI-251
- AI-252
- Task group proposal (new AI-266)

Joyce Tokar:

- AI-249
- AI-265
- AI-266

New Action Items

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

Steve Baird:

- AI-167
- AI-216

John Barnes

- AI-254

Randy Brukardt:

- AI-85
- AI-224
- AI-246
- AI-248
- AI-260

Editorial corrections only:

- AI-161

- AI-195
- AI-225
- AI-229
- AI-233
- AI-238
- AI-240
- AI-257
- AI-267
- Object_Size attribute

Gary Dismukes:

- AI-158
- AI-273
- Investigate the incompatibility impact of the proposed resolution of AI-230 (see details below).

Bob Duff:

- AI-239
- AI-259
- Be the test creator of last resort
- Fix limited types a bit to permit aggregates and initialization by function call [jointly with Tuck]

Mike Kamrad:

- Amendment on Assert pragma
- Various items to be standardized [jointly with Mike Yoder]
    - Discard_name & ‘image
    - External_tag
    - Storage_IO of tagged types
    - Array indexed by holey enumeration
    - Static elaboration
    - GNAT attributes and pragmas
- CPU time (separate from real-time) [jointly with Joyce]

Pascal Leroy:

- AI-228
- AI-264
- AI-272
- Make a proposal for non-reserved keywords.

Steve Michell:

- AI-148
- AI-250

Erhard Ploedereder:

- AI-147
- AI-237
- AI-277

Tucker Taft:

- AI-133
- AI-162
- AI-188
- AI-214
- AI-217 (on hold pending abstract package proposal)
- AI-230
- AI-231
- AI-251
- AI-252
- AI-266 (Task group proposal)
- More stream representation control (split from AI-195, now AI-270)
- User defined assignment/subtype conversion
- Physical units (length/dimensional analysis), whereby square inch units are generated by result of multiplying inches; subtypes with special attributes would be used to provide the specifications of these dimensions, reducing the introductions of lots of extra operators.
- Fix limited types a bit to permit aggregates and initialization by function call [jointly with Bob]
- Abstract package proposal

Joyce Tokar

- AI-249
- CPU time (separate from real-time) [jointly with Mike Kamrad]

Mike Yoder:

- Various items to be standardized [jointly with Mike Kamrad]
    - Discard_name & ‘image
    - External_tag
    - Storage_IO of tagged types
    - Array indexed by holey enumeration

- Static elaboration
- GNAT attributes and pragmas

All:

- Recommend new members. John: completed his.

WG9 Actions

Steve Baird has been proposed by Pascal Leroy to be new member. He is expected to be officially nominated at the next WG9 meeting in Vienna, Austria, so until then he is attending the meetings as an observer. It was pointed out that more user representation is needed in the ARG; currently Mike Kamrad is the only user. Tuck suggested that Bill Pritchett from DSC Corporation be approached as Bill gave an impressive presentation how successful Ada is in Vetronics at the recent SIGAda meeting. Mike Yoder would also be a good candidate, assuming he is interested.

Pascal pointed out the official ISO process to add new members is obscure. When Jim Moore was asked he pointed to the national body nomination rule. Consequently, Pascal recommended waiting on Steve's nomination until USTAG approves.

Pascal mentioned that France had suggested at the WG9 meeting that National Bodies should filter and submit proposals. It is noted that proposals submitted formally by a National Body must have a formal response within a fairly short time. Thus such proposals would increase, rather than decrease, the workload of the ARG, and would make it much harder for the ARG to set priorities for amendment work. The point was made that the relationship between National Bodies and the ARG would be more productive if it worked the other way around, i.e. if the ARG (or WG9) "subcontracted" studies of difficult topics to National Bodies. This is what happened when the study of the adaptation of the vector/matrix packages to Ada 95 was subcontracted to the National Body of UK, and it proved very fruitful.

The ARG concludes that it is best to encourage National Bodies to make informal comments about amendment proposals, but to discourage formal amendent proposals from National Bodies.

ARG Procedures:

Randy reported on the changes he made to the procedures since the last meeting, including:

- Cleaning up the format of the document (which now is a PDF document).
- Lots of minor editing, such replacing the term "Chair" by "Rapporteur".
- Lots of work on the handling of commentary coordination among Editor and Rapporteur and other ARG members to determine if a submitted commentary is to receive action or not, namely to be put onto the ARG meeting agenda. ARG members can get a commentary placed onto the agenda with the approval by 10% (or at least 2 members) of ARG membership.

There was no substantial discussion on the procedures and they were approved, 9-0-0.

Detailed Review

The minutes for the detailed review of AIs are divided into existing amendment AIs and non-amendment AIs. The AIs are presented in numeric order, which is not necessarily the order in which they were discussed.

Detailed Review of Amendment AIs

AI-216: Unchecked Unions -- Variant Records With No Run-Time Discriminant

Tuck reported that the only substantial change was limited to removing the possibility of giving a component clause for a discriminant in a record representation clause. The rest of the changes were editorial.

John points out more editorial nits about extra ‘s’ in the text.

Steve Baird points out there is a missing rule in 3rd from last paragraph under discussion concerning the use of an Unchecked Union as a completion of a private type; in such a case, the private type must not have known discriminants. (And there has to be a similar rule for generic formal private types.) Note that a private type is never an Unchecked Union, even if its completion is.

The group then discusses whether Unchecked_Union is a representation pragma or not. It seems that it should be, because it affects the representation of a type. But this implies that, when the pragma applies to a full type, according to the existing legality rules it applies to the partial view, and exposes more privacy than necessary. This began a discussion of how the pragma interacts with generics. A proposal is made (similar to the restrictions for private types, above) with respect to generic derived types, namely, if the actual type is an Unchecked Union, the formal must not have known discriminants. Alternatively, the following concept was proposed as simpler wording: in a generic instantiation, if the actual type in an Unchecked_Union, the formal type must not have known discriminants or it must also be an Unchecked_Union. Again, a similar rule is needed for private extensions: if the full view is an Unchecked_Union, either the partial view must be an Unchecked_Union, or it must not have known discriminants.

Gary expresses concern that we may need a similar rule for incomplete types. This is not a problem, as it is impossible to access the discriminants of an incomplete type.

Tuck produced the following wording to cover this problem. It replaces the 4th from last paragraph under discussion.

> “A private type is never an unchecked union (even if its full type is). As usual a derived type inherits the unchecked union representation aspect from its parent type. A formal derived type is an unchecked union if its specified ancestor is an unchecked union.
>
> If an unchecked union completes a private type or private extension, the partial view must not have known discriminants or it must be an unchecked union. For contract model reasons, in an instantiation of a generic, if the actual type is an unchecked union, the formal type must not have known discriminants, or it must be an unchecked union.”

Vote on intent 8-0-1. Steve will complete the final changes and produce the detailed wording.

(Note: There was some confusion over version numbers on this AI from Tuck. It reminded us that version numbers are valid only when Randy enters them on the AI.)

AI-230: Generalized use of anonymous access types

Gary wants to know why this AI is important. Erhard says that the object-oriented community expects that a pointer to a type in a class is convertible to a pointer to a class-wide type. Gary and Pascal are concerned about the size of the language change for this proposal.

Tuck described what he changed in the AI since the Leuven meeting:

- The accessibility level was defined to be associated with the type, not the object; consequently:
    a. For a subtype of an anonymous access type, the accessibility level is the level of the subtype declaration.
    b. For a constant declaration, the level is the same as the level of the type of the initial value.
    c. For a function return, the level is the same as the level of the type of the returned value.
- The conceptual storage pool is associated with the accessibility level; therefore, all anonymous types at the same accessibility level share the same default storage pool.

What happens when type conversions between anonymous types and deallocation are involved? Subtypes of anonymous types can be used as a mechanism to “unchecked deallocate” objects of the anonymous access type.

Erhard proposes that if two access types at the same accessibility level are convertible then they use the same storage pool.

Tuck suggests restricting named subtypes of anonymous access types by not permitting allocation and deallocation of objects of those named access subtypes. Named subtypes were his method of handling components of anonymous types, but if the accessibility level of a component is that of the enclosing object then named subtypes are not needed. It seems that named subtypes of anonymous access types cause a lot of trouble, so maybe they should be eliminated from the proposal.

After Erhard mentions that having named subtypes is useful for explicit type conversion, Tuck suggests a different approach: allow implicit conversion of any "access all T2" to "access all T1'Class", where T2 is covered by T1. This is a simple alternative to the existing AI: it means that conversions that require no checking are implicit. This is not upward compatible, as it could cause existing programs to become ambiguous. There is a lot of discussion on whether this incompatibility is acceptable or not, and as usual this discussion leads nowhere in the absence of hard data. Gary offers to test how upward incompatible this change would be with GNAT. Tuck will create an alternative AI.

Meanwhile, he will update the original AI so as to properly handle components and he will eliminate named subtypes of anonymous access types.

AI-231: Access-to-constant parameters and null-excluding subtypes

Gary wants to know the rationale for this AI. The rationale is found in the discussion section.

Tuck explains that the changes since the Leuven meeting were to formalize and simplify the previous proposal.

Pascal says that this proposal exposes the fact that the "access T" syntax for anonymous access types is very odd, as it looks a lot like a pool-specific access type, and there is no indication that the parameter cannot be null . Yet his counterproposal of making the "not null" explicit or the introduction of configuration pragmas seems clumsy. The current proposal very succinctly describes the semantics of "access T" as equivalent to "access all T not null".

Pascal finds the "not null" construct awkward; he suggests adding the reserved word is to make the syntax phrase "is not null". This suggestion finds no support; it is felt that two "is" in a type declaration is ugly.

Tuck is directed to add as justification for this AI the improvement in safety and performance of software.

Steve notes that default initial values raise Constraint_Error for a "not null" type. 'Input would raise Constraint_Error if a type contains a component with a default initial value. Obviously, "not null" components always need an explicit initial value.

It is interesting to notice that we can now have access subtypes which are constrained in two steps:

```
type A is access String;
subtype B is A (1 .. 10);
subtype C is B not null;
```

This may not be problematic, but it will probably require some care in wording the proposal.

Steve also points out that the rules about statically matching constraints need to be defined for "not null" constraints. Also the AI needs to cover generic access types: we want to require exact matching of the actual with the formal. And real wording for 8.6(25) is missing.

Finally, a syntactic ambiguity with the use of the "not null" constraint in a named access type definition is uncovered. Consider:

```
type A is access access B not null;
```

Does “not null” apply to the anonymous access type “access B” or to the named access type A. Tuck would not permit “not null” constraints for subtype indications; he wants to limit” not null” to subtype marks only.

Vote on intent: 7-0-2. Tuck to revise.

AI-248: Directory operations

Once again the motivation for this AI is questioned. Users want it in the standard library and with the Ada POSIX interface in jeopardy, it seems to be necessary.

While the discussion of this AI was scattered among the operations and types of the package specification, the minutes have organized the discussion and decisions by how the operations and types were listed:

Create_Path (formerly Create_Tree):

The name Create_Path better describes the intent of this operation than Create_Tree. A recommendation to add a Boolean parameter for recursion instead of having a separate routine was defeated by a 3-2-5 straw vote. Create the directory with name New_Directory with any necessary enclosing directories. Use the same wording that is found in A.8.2 for files for the raising of exceptions. The last sentence needs to be changed to refer to “any directories” instead of a single directory. The description also should say “zero or more”, as this routine does not have to create something.

Current_Directory:

Is the default directory defined? Not really. Default directory is useful for the use of simple names for directories and files. What it should be is what would be expected from the underlying environment. But Windows has a default directory for each device, which is very different from Unix. Then it should return a full name. Also use the same wording that is found in A.8.2 for files for the raising of exceptions.

Set_Directory:

Accept simple names (using the default directory as the enclosing directory) or full name.

It was during the discussion of this operation that it was determined the terms simple names, full names and default directory need better definitions before the description of operations. (It was noted that portability is best supported through the use of simple names.)

Delete_Tree (formerly Remove_Tree):

Correct the cut-and-paste error in this description. Use_Error exception is now raised if it is not able to delete the entire tree (or if that tree does not exist). It is noted that when Use_Error is raised, it is possible that some files and/or directories are deleted. A recommendation to add a Boolean parameter for recursion instead of having a separate routine was defeated by a 2-5-3 straw vote.

At this point the question was raised as to why use the name Remove_Tree instead of Delete_Tree? The reason given is that most operating systems use the term remove. But remove is inconsistent with the term delete for files in the current language. It was decided to use the term delete, so names Delete_File, Delete_Directory and Delete_Tree will replace Delete (file), Remove_Directory and Remove_Tree.

Containing_Directory (formerly Parent_Directory):

It was decided to change the name to the new name as the term containing directory is more descriptive of what is being requested. This function will return a full name of the directory of the containing directory.

Rename:

It was recommended that the Implementation Advice say the Rename function should work when both names are simple names.

The discussion brought up the lack of copy operation for files or directories. It was decided to add a Copy_File operation, but no copy directory. Pascal cautions about the difficulties with special files on Unix (and similar beasts on other operating systems). Tuck argues that this shouldn't be a problem for files created by Ada. Copy_File should be defined to copy files that can be created by Ada packages, and to have an implementation-defined behavior for other files.

The discussion turned its attention to the search operations and supporting types, leaving the discussion of full and simple names for later.

Is_Valid:

The need for this operation is questioned. Randy says that it is a useful test for determining if a search produced something meaningful and Bob supported his point. Tuck pointed out that we have the constant No_Directory_Entry for testing of the result of a search, and therefore that operation Is_Valid appears to be redundant. If Is_Valid is retained it should mean the same thing as comparing with No_Directory_Entry. The straw vote on this meaning for Is_Valid was approved, 6-0-4, but the group decided that the redundancy was unpleasant and approved removing Is_Valid by a 7-1-2 straw vote. Note: later in the discussion it was restored!

Start_Search and End_Search:

The Start_Search operation is an iterator operation on the search object. Consequently Search_Type should be a controlled type to properly support the finalization of search objects.

There was a discussion on whether a call on End_Search should be optional. If this is the case, then it means that the finalization of the search object (by leaving the scope of the object, or by starting a new search with the same object) has the effect of calling End_Search. The End_Search subprogram should be retained as a way to release operating system facilities early (i.e., before leaving the enclosing block). The meeting voted to make calls on End_Search optional by a 5-2-3 straw vote.

Then there was a recommendation that Start_Search act as a restart, thereby eliminating the need to explicitly call the End_Search operation to restart a search. The group approved making Start_Search act as a restart by a 6-2-2 straw vote.

Get_Next_Match:

The only significant issue on the Get_Next_Match operation was how it handled an unstarted search object: either return No_Directory_Entry in the Directory_Entry parameter or raise a Status_Error exception. The exception approach is similar to how the existing file operations handle unopened files and was approved by 6-3-1. At the end of the iterator, Get_Next_Match returns No_Directory_Entry.

Is_Directory and Is_Ordinary_File:

The motivation for distinguishing ordinary files is due to the existence of special files, such as symbolic links to existing files. It was suggested that a more direct way of distinguishing among directories, ordinary files and special files was by defining an enumeration type with these values, such as:

```
type Directory_Entry_Kind is (Directory, Ordinary_File, Special_File);
```

and then change the filter type to be:

```
type Filter_Type is array (Directory_Entry_Kind) of Boolean;
```

Consequently the two functions Is_Directory and Is_Ordinary_File are replaced with a new function Directory_Kind that operates on directory entry objects and returns the new enumeration type. Also the filter parameter in the Start_Search operation is changed.

Special files appear to be files that Ada programs can't create or read. Bob believes there are lots of programs that would like to read soft link files in order to find the target. If that is the case may be the best way to handle special files is to provide implementation advice but what is that advice? Does POSIX provide any directions? Not really because it is allowed to deal with all files types that this package doesn't want to handle. It was decided the specifics of special files, such as soft links, should be left to AARM where operating system details can be discussed.

Composition of simple and full file names:

Randy explained that he borrowed wording for full names from the RM for the Name operation. He further noted that he did consider Jean-Pierre Rosen's documentation on composition, where he divided the composition of file names into these elements:

- Device
- Path
- File
- Extension
- Version

This decomposition of file names is meant to be the union of all operating systems. There is a risk that another operating system does not fit this composition. Consequently, he decided against adding any operations to decompose full names. It was noted this list was missing a network element (the host name).

Should all of this be avoided by putting the details into a child package? Tuck and Erhard argue that composition of full names for files and directories should be available in this package. Pascal would like to have some access to the composition elements without doing string manipulations to get them from the name strings.

The focus is narrowed to Simple_Name function and the composition functions. Tuck argues for just one composition function for either directories or files and let an implementation do the right thing. Portability between systems and implementation is the goal of this interface and that makes the system-specific elements, beyond file and directory names and extensions, difficult to expose.

After much discussion, it was decided to take a series of straw votes on how to proceed:

- On whether this package should provide all three kinds of operations, namely compose full file and directory names, extract relative names (the containing directory name or the simple name) and file extension operations (extract simple name or extension and compose file name with simple name and extension), it was defeated by 3-3-4. John commented that he didn't like the notion of making "extension" a first-class citizen.

- The group took a step back to see if there would be agreement on just the compose and relative name operations and it did by 8-0-2 straw vote.

- After taking another straw vote to support more operations (6-3-1), the group supported the addition of just the extension operations (i.e., Base, Extension and Compose_With_Extension) by 5-2-3.

Erhard would like to have a (modification) time stamp and size operations. Randy says it is difficult to know what size should be returned, especially in the presence of 64 bit values and to know how to handle time stamps for operating systems that do not produce time stamps. For those operating system with no time stamping, a time stamp operation would raise Use_Error. Consequently the meeting approved time stamp (meaning the time when the file or directory was last modified) by consensus.

As for the size of files, it was suggested to let the existing Count type in Stream_IO package (A.12.1) handle this problem. Unfortunately its bounds may not be appropriate to deal with very large files. It was decided not to change the type of count in A.12.1 to be an implementation-defined integer type. Instead a new signed type should be declared in Directory_Operations; the upper bound is implementation-defined and the Size function returns values of this type or raises Constraint_Error if the size is an illegal value for the implementation. The reason why the group eventually decided on a *signed* type is because operations on Size values are common, and we don't want modular semantics for them. It is implementation defined what the Size function produces for directories or special files.

All query functions, such as Kind, Modification_Time or Size, should be applicable to both names and Directory_Entry_Type values.

The term “same containing directory” should replace “same directory location” in the Implementation Advice on Rename paragraph and the AARM ramification paragraph should be dropped.

Finally the discussion ended with debating whether Directory_Entry_Type should be a limited type, since there appears to be no need for its assignment. This would simplify the implementation and it would be treated similarly to how files are treated. If assignment is needed, then the program should store a pointer to it. Consequently this means the constant No_Directory_entry is no longer needed and the Is_Valid function is restored!

Vote on Intent for all these changes: 7-1-1.

AI-249: Ravenscar Profile for High-Integrity Systems

The HRG meeting at SIGAda produced four additional changes to version 3 of the AI:

1. Keep FIFO_Within_Priorities as the default Task_Dispatching_Policy and Ceiling_Locking as the default Locking_Policy but add to the discussion that another profile can be defined that assumes different policies.

2. Drop Max_Asynchronous_Select_Nesting restriction because the restriction No_Select_Statements covers this restriction.

3. The rules for eliminating blocking in protected actions (as found in paragraph 1.4 of the AI) should be encapsulated in a new restriction to be added to the Ravenscar profile.

4. A No_Task_Termination restriction should be added that permits an implementation to define how violations of this restriction will be handled (i.e., what happens if termination actually takes place).

The ARG approved change 1 by 5-0-4 and change 2 by 9-0-0.

The ARG recommended a new restriction for change 3, No_Blocking_In_Protected_Action, with the semantics that Program_Error must be raised when blocking is detected in a protected action. Approved 9-0-0.

The ARG recommended that the new restriction in change 4 be named No_Task_Termination. Approved 9-0-0.

Furthermore the ARG recommended that the AI be split into two AIs:

1. Define what the profile means with respect to restrictions and define the format for the profile.

2. Define the additional new restrictions that currently constitute the Ravenscar profile.

This recommendation as well as the intent of changes 1-4 above was approved 9-0-0.

Joyce is directed to do the split and produce recommended wording. (If Mike is available he can be her backup.)

AI-251: Tagged Types, Abstract Interface, Multiple Inheritance

Tucker enumerated the changes he made to this AI since the Leuven meeting:

- Added the problem statement
- Added Tag attribute on interfaces for the membership tests
- Allowed interface as actual when formal were private extension of interfaces; limited the instances of interfaces where interfaces could be passed to tagged type, meaning abstract tagged types
- Enriched and organized discussion section into four parts:
    - Possible Implementation Model

    - "Thin Pointer" Implementation Model
    - Shared Generics
    - Possible "is null" rather than "is abstract" for primitives
- Added more details on "is null" as a placeholder for concrete null implementation (see note)

Tucker reviewed the implementation model once again and then described the addition of a "directory" associated with the implementation whose purpose is to perform interface-to-interface conversion. The directory could be implemented as a hash table. The location of this directory could be with the implementing type's dispatch table or with the implementing type's interface table (like Java does).

The "thin pointer" implementation will use the same directory to produce the interface view necessary for parameter passing. The cost of the directory search can be minimized by saving the result and by reusing it for other references. Otherwise the combination of fat/thin pointers is similar to how access-to-unconstrained-arrays are handled.

Tucker then explained the details of the additional feature, "is null". The effect of declaring a primitive of an interface to be "is null" is a body with only a null statement. The purpose of the feature is to enable certain abstract types that have null (non-abstract) default implementations of certain primitives to also be converted to being abstract interface types:

```
generic
   type N is abstract interface with private;
package P is
   type NN is new N;
   -- actual for N will have concrete (if null) implementation
   ...
end P;
```

This allows non-dispatching calls on the primitive of the generic formal type to be made inside the generic which may be necessary to enable a pass-the-buck-to-parent paradigm, when overriding the operation. To get the parent operation in converting to the actual interface for N, the actual interface better have a concrete (if null) implementation.

Steve asked about inheritance of interfaces in private parts, as shown by this example:

```
package P is
   type T1 is tagged private;
   ...
private
   type T1 is new Foo and I with ...
end P;

...type T2 is new P1.T1 and I with ...;
   -- T1 already implements interface I?
```

Now T2 inherits I *only once*, and it could override the operations that come from the private parts of T1 (because there is only one slot for the operations inherited from I). This has the effect of making the private part visible to everyone. Not good. The existence of fat pointers everywhere (no thin pointers) could possibly allow implementation of such private interfaces but the cost would be too high.

Bob proposes to make the existence of private interfaces illegal; if a type implements an interface then it must specify this in the public part. This situation is analogous to prevention of abstract operation in the private part. Of course, making the implementation of interfaces public has a ripple effect on all derivations of the type: if, during maintenance, someone decides that the type implements another interface, then all the descendants of that type have to be updated to implement the new operations. Even so, it was decided that the rule that prevents "secret interfaces" is the only way to go, as we don't want to break the contract model of private types.

Pascal still does not like the fact that “interface” does not appear in the syntax to reinforce significance of the interface specification. There is a desire not to add another keyword. Instead it was suggested that a special class of non-reserved keywords be created so that only the use of “interface” in the declaration of the abstract interface has keyword significance. A similar mechanism could be used in other amendments if needed, in order to avoid odd combination of existing reserved words.

The discussion finished with a series of straw votes on these issues:

- Putting “interface” into the abstract interface declaration was approved, 8-1-1; then the vote to make “interface” a reserved word was not strongly supported, 4-4-2. Pascal volunteered to do the necessary wording and syntax for the special class of non-reserved keywords. This will go into a new amendment AI.

- The meeting approved keeping “abstract” in the root type declaration (5-1-4), in the type derivation declaration (6-1-3) and in primitive operation declaration (8-0-2).

- The “is null” proposal was approved, 6-0-4.

The discussion finished with approving the intent of the changes to the AI, 9-0-1. Tucker is asked to modify this AI quickly.

AI-252: Object.Operation Notation

Why do we need this feature?

Proponents say it eliminates the need for using the package name for the operations, which is an annoyance to many coming from other OOP languages. It could be more attractive to newcomers.

Those opposed say is that you can get around this problem with use clauses. Can we afford the ARG time and implementation expense for this feature?

A straw vote to continue work on this AI passes 6-3-1.

Various simplifications are suggested. Erhard suggests not worrying about named access types. Tucker wonders if restricting this feature to just composite types would help. Erhard suggests going further and restricting it to tagged types. Does disallowing elementary and arrays as the prefix help? (That is, the prefix would be restricted to things for which dot already is allowed.) The group expresses lukewarm approval for this improvement.

Bob worries that private completed by integer could use this notation outside the package, but not inside, which seems weird. It is left to Tucker to come up with a new proposal.

AI-254: Downward Closure

John chose the anonymous access-to-subprogram approach over limited access types approach because it is more familiar and similar to how other languages, like Pascal, handle this problem.

Then the second of two questions, “How to do it?” was addressed.

There will no type conversion, and no .all’Access (this will be achieved by saying that the accessibility level of these types is infinite). Tuck asks why we want to forbid conversions, since we need to carry around static links/display anyway. But Randy explains that a simple accessibility test doesn’t work because these accesses can be passed to a scope that has same level.

The shared generic complexities convince Tuck that type conversions aren’t necessary. Separate it from the call back paradigm. Eliminating conversions also means that there is no need to pass dynamic accessibility levels with parameters.

Bob proposes to drop access from parameter description and the 'Access on the actual call; this is similar to how Pascal does it and it does separate it from the callback mechanism. In other words, we would have a subprogram type, not an access-to-subprogram type. Pascal points out the ambiguity problems with parameterless functions, where you would not know if you are passing the subprogram value or the result of the call. Existing subprogram access value would need to use an explicit .all to be passed to subprograms with subprogram parameters.

Approved as written with explicit use of access versus the subprogram values: 5-1-4.

Then the other question, "Is this feature needed?" was addressed.

Gary and Pascal stated that there is too much complexity in this feature since the user need has not been clearly demonstrated. Bob counters that there's lots of use of 'Unrestricted_Access (for users of GNAT) to prove the need. Erhard says that block structured languages need this capability but deeply nesting subprograms is a thing of the past. Bob counters that the use of iterators provides a compelling example without going beyond two levels of nesting.

The group approved the continuation of this AI, 5-2-3.

The discussion then turned to some other details of the AI. The clumsiness of the syntax when the subprogram description also has access subprogram type is the price to be paid to keep from the problems of naming these types.

Access discriminants are distinctly different from subprograms-as-parameters and therefore use of the new types in discriminants is going to be disallowed by the AI.

Does the AI include protected operations as access values passed to these parameters? From a static view it does not seem too much new work and is consistent with how named access types deal with protected operations. But the runtime costs and compelling examples are unknown. This was defeated by a vote, 1-1-8.

Anonymous subprogram access parameters are "not null" subtypes; we do not want the actual to be null. We also should allow default parameters in the signature; in this way, these are similar to renames.

AI-257: Restrictions for implementation-defined entities

There is a discussion of the wording needed to make the restriction not apply to partitions. 13.12(8) allows a restriction to not be required to apply to the entire partition, but there are no examples of such restrictions in the standard.

Tucker suggests changing the wording to say: "These restrictions apply to the compilation units in the compilation where the restriction pragma appears; if the compilation contains only the pragma…"

Pascal suggests changing the wording to say: "The partition need not obey the restriction even if this restriction applies to some compilation unit included in the partition (see 13.12)."

Pascal's suggestion is adopted.

Add a !corrigendum section.

Approved with editorial changes: 9-0-0

AI-260: How to replace S'Class'Input with custom versions

Change the title: How to control the tag representation in a stream.

Randy opines that this is just the tip of a wider problem (the inability to write a function like S'Class'Input without knowing all of the possible result types). Tucker had proposed some ways to solve that problem during lunch. Shouldn't we address the wider problem instead of a specific solution? The group thinks that this is useful even if the wider problem is solved. Using the proposed attributes would be much easier than rewriting S'Class'Input, and these attributes are fairly easy to implement.

These are operational attributes with all of the effects that this implies. They have similar inheritance rules as ‘Read and ‘Write have. (But this was changed later in the discussion!)

Correct the wording: Formal S → Stream. Item’Tag → Tag

Tucker now states that he thinks that his previous opinion was wrong; these attributes should be class-wide only, without inheritance. Much discussion ensues. Eventually the group concludes that the attributes make the most sense only on the class-wide type, as they are only used in dispatching operations.

No inheritance would mean that child types don’t get it. If T2 is derived from a type T1 which had a user-defined Tag_Write, T2'Class’Output wouldn’t use the user-defined Tag_Write. This seems wrong. As a solution to this problem, Tucker suggests that the default implementation should be a call to the Parent’Class operation. This is conceptually similar to inheriting them.

Someone makes a more radical suggestion to disallow overriding of these attributes. There is not much support for this idea — it might be useful.

In summary: the attributes are class-wide, inherited (by the default operation calling the parent), and there is no restriction on overriding. Approved this intent: 6-0-3

AI-266: Task Termination procedure

Tucker explains the task group proposal. Three routines are defined that are called after task waiting and finalization, but before Ids and task attributes go away. The task itself calls these. That is necessary to pass the exception occurrence.

The question is raised as to whether these are called if the task fails during activation. Doing so would make it impossible for the task itself to call the routines. No single answer emerges from the discussion, many different opinions are voiced.

Someone suggests that we might need a fourth routine, to be called when a task goes away because it is waiting on a terminate alternative. This sounds a lot like normal termination, so the idea is not pursued any further.

Task groups are a property of an object, and it is possible to force components to be in the same task group.

Pascal suggests that the task group is an operational item, not a representation item. He continues that they should be therefore an attribute, as we don't have operational pragmas. Tucker resists, believing that the group should be a representation item.

Randy points out that a representation item can’t be overridden for a derived type, while an operational item can. That matches the proposed semantics of task groups pretty well. Tucker finally gives in. Pascal says we could have an operational pragma; it wouldn't be too difficult to define.

Bob says this is ugly; the implementation would need to make a system call on all of the components to support using a non-default task group for the entire object.

Tucker thinks that we should say that the task group object has to be elaborated before the task object, otherwise we could be using an unelaborated object.

Do we need this mechanism? Ravenscar does not need this mechanism, as it makes task termination implementation-defined. The requirement came from the Exception Workshop at Ada Europe 2001. It was felt that it is embarrassing that tasks silently disappear in Ada.

Erhard suggests that the task group be attached to masters, not to scopes. At the creation of a task, that task would get the task group of its master by default.

Tucker comments that then you might need an extra master (if neither finalization nor tasks are used in the scope). This doesn't seem too bad.

Someone asks how to assign a task group to use for all tasks in the entire partition. This isn't possible because of the task group object. It has to be elaborated before it is used, but of course the environment task starts before any objects are elaborated. It would be possible to put it in the root of a parent to most of the system.

Suggestion: Add a runtime call for establishing the default task group; it would be retroactive (all existing tasks). (Tasks that died before the runtime call would not call the task group object.)

Bob notes that the pragma does not affect the enclosing master for a package. (Consider a nested package.) Probably would have to set it and reset it on the entry and exit to the elaboration code for a nested package.

Should there be restrictions on the overriding procedures? Probably, but complex routines would be unusual. (Note that Finalize has similar issues, yet no restrictions were placed on it.)

Someone suggests limiting the application of the pragmas to limited types. But that wouldn't allow them to be used on access types.

The consensus is that this is too complex a solution for the problem. Tucker should look for simpler solutions.

AI-267: Fast float to integer conversions

Add a !corrigendum section.

Tucker would like a note like paragraph 2-4 of the discussion added to the standard. The group feels that such a note would clutter up the manual. It would be useful to include that in the AARM.

Bob notes that “Float type” should be changed to “Floating point type”.

Approved with editorial changes: 8-0-1

AI-277: Handling mutually recursive types via separate incomplete types

Tuck does not like the syntax of Randy’s proposal because it further complicates type definition. Instead, Tuck likes it if the completion of this mutual dependency is a subtype declaration, such as:

```
package P is
   subtype T1 is separate;
   ...
end P;

package Q is
   type T2 is <full type>;
   separate subtype P.T1 is T2;
end Q;
```

Pascal counters with this alternative syntax

```
package P is
   type T is separate;
   ...
end P;

package Q is
   separate (P)
   type T is ...;
   --  operations of T
end Q;
```

The appeal of Tuck's approach over Randy's or Pascal's is that it limits the separateness to subtype rules as opposed to complicating type rules.

Pascal is concerned that Tuck's proposal introduces a whole new concept at this point and it appears to be totally different from incomplete types. He would prefer the abstract package concept or Randy's proposal over Tuck's proposal. Also, he doesn't like the fact that the separate (sub)type and its completion may have different names, which doesn't improve readability of the code. Since both approaches avoid the problems in the with type proposal, Erhard points out that the real difference appears to be only style.

Tuck modifies his proposal to make T1 an incomplete type and leave the completion as a subtype.

Erhard is bothered by the fact that the package where the completion is found is not designated. Such an indication is useful to both the reader and the compiler. This could be specified as

```
type T is separate in <fully qualified package name>;
```

Tuck recommends that completion package name be not a renamed package.

When both packages are visible, then Tuck would like to able to use operations on the separate types. (Note all of this visibility and semantic dependency issues were all discussed in the with type proposal.) It is the indirect dependency that causes problems (for all approaches) as this example illustrates:

```
package P is
   type T is separate ...;
   ...
end P;

package Q is
   type T is ...; -- Completion
   ...
end Q;

with P, Q;
package R is
   type A is access P.T;
   X: A;
end R;

with R;
package S is
   ... R.X.all  -- illegal without importing Q
...
end S;
```

The issue is whether a separate type is considered incomplete when you can only see its completion indirectly. Tuck recommends that you cannot operate on an object of a separate type unless you with the "completer" package. Here visibility rules over semantic dependency.

Tuck points out you may reference a function that returns the "incomplete" type and calling that function is OK even though you only with the package with the incomplete type (that's an expression of an incomplete type, which is a new thing). Dereference of an access to an object of a separate type is not OK, though, because dereferencing an access to incomplete type is always illegal. Tuck further states that when you declare an object (and other things that freeze the type or require it to be completed) you have to with the "completer" package.

Steve points out an odd implication of this last point, with this example

```
package P is
   type T1 is separate in Q;
   ...
end P;
```

```
package Q is
   type T1 is ...;
   ...
end Q;

with Q;
package R is
   subtype S is Q.T1;
   ...
end R;

with R, P;
package S is
   X: R.S;  -- legal
   Y: P.T1;  -- illegal, which appears odd, but is fine
   ...
end S;
```

Erhard extends his proposal to eliminate all explicit completion, that is, the completion doesn't have to specify which type it completes. He notes that this allows directly declaring mutually dependent types without any extra packages at all:

```
package Employees is
   type Department is separate in Departments;
   type Employee is record...
   ...
end Employees;

package Departments is
   type Employee is separate in Employees;
   type Department is record...
   ...
end Departments;
```

The intent of Erhard proposal is approved, 8-0-1. Erhard will do the rewrite as a separate, new alternative AI.

On Sunday, Erhard asks the group about the interaction of this AI with access types? (Most such uses require an access type.) Pascal and Randy suggest that if there is only one access type, then it should be declared after the incomplete type stub.

Erhard points out that you can't do that if you use just an incomplete type stub in a pair of packages, you'll need a separate interface package. The group doesn't find that to be a major concern, most of the other proposals required that as well. The fact that Erhard's proposal doesn't always require an extra package is a bonus, not a requirement.

Detailed Review of Regular AIs

AI-85: Append_File, Reset and positioning

The AI should explicitly state that whether a file supports positioning or not may depend on the mode. (In particular, a file may not support positioning for the Append_File mode, but still support positioning for the In_File and Out_File modes.)

Approved intent: 7-0-1. Randy will edit.

AI-147: Optimization of Controlled Types

Tucker's proposal was that Initialize/Finalize pairs can be eliminated if it is known there is no user-defined Initialize routine. As previously, this permission would apply only to non-limited controlled types.

The suggested change to the 4th paragraph of the summary is:

> This permission does not apply to user-defined Initialize…

Approved intent: 7-0-1. Erhard will revise and add the wording.

AI-161: Default-initialized objects

A !corrigendum section is needed.

Approve with change: 7-0-1.

AI-167: Scalar unchecked conversion is never erroneous

Steve notes that the Rational compiler tries to insure that values aren't invalid, so that this change would push Unchecked_Conversion in the "expensive" category by requiring checks after any use of it.

Tuck agrees that the same problem would probably happen in their compiler when assigning into a default-initialized value. Such an Unchecked_Conversion would have to be checked at that point.

The group decided that changing any Unchecked_Conversions to be non-erroneous would have unacceptable performance implications. A search for alternatives was started.

Steve suggests that renaming the result would provide a solution:

```
declare
   A : Integer renames UC(123.0);
begin
   if A'Valid then ...
end;
```

This solution was not received well, as it is certainly not obvious to the average user (or to the average ARG member for that matter, since this topic has been discussed several times already, and it is the first time that this suggestion is made).

Providing an attribute that does unchecked conversion and includes the validity check was suggested. This was shot down because all existing attributes take a specific type for their arguments.

A generic that tests the result of an unchecked conversion for validity is proposed:

```
generic
   type Source is private;
   type Target is private;
function Unchecked_Valid (A : Source) return Boolean;
```

Should the Target type be discrete? It really ought to be elementary, but we don't have an elementary formal type. It should be definite.

Concern was expressed about the cost of doing the conversion twice (once to check validity, and once to do the actual conversion). A generic that does both operations together is proposed:

```
generic
   type Source (<>) is limited private;
   type Target is private;
procedure Is_Valid_As (S : in Source;
                       T : in out Target;
                       Valid : out Boolean);
```

S is Unchecked_Converted to Target. If the result is valid, Valid is set to True and T is set to the result of the Unchecked_Conversion. Otherwise, Valid is set to False and T is not modified.

The rules of Unchecked_Conversions apply to Is_Valid_As, as it implies an Unchecked_Conversion.

This specification implies that this is defined for record types. A long discussion ensues as to whether it should be allowed for a record type. This is problematical for all but the simplest records. For instance, should gaps need to be checked? Tucker says yes, if the implementation assumes something about them. Pascal suggests that we should only support this generic for scalar types.

The group concludes that it must work for scalar types, and should be implementation-defined for other types. Implementations may reject the instantiation for non-scalar types. There should be implementation advice that it should be supported for types that have no access subcomponents, no gaps, no implicit components, no discriminant-dependent components, and a contiguous representation.

Proposal: Unchecked_Conversion should remain erroneous as it currently is. A new generic should be provided.

Approved intent of the above proposal: 6-1-1. (John opposes, would like an attribute to do the check rather than a generic.)

Steve Baird will write up this new proposal.

AI-195: Streams

Pascal describes the changes, mostly to sections 2 and 3. 3 fixes bugs in the corrigendum.

Bob suggests that we need to add a sentence in chapter 8 to say that clauses have a visibility. This amounts to saying "declaration or clause" instead of "declaration" at the right place.

A suggestion is made to change the last sentence of 13.13.2(9/1) to say that the check is done at the freezing point.

John: Need a space after the dash in paragraph 7 of the summary.

Fix example to complete tagged type with something other than Boolean.

Kiyoshi: Paragraph after example: "NNT is not callable" → "NNT'Read is not callable".

Approved with editorial changes: 6-0-2

AI-225: Aliased current instance for limited types

Typo in paragraph 1 of question: "it's" → "its".

Wording: take out parentheses around "(full)".

Approved with editorial changes: 8-0-0.

AI-228: Premature use of "must be overridden" subprograms

Assigned to Pascal Leroy.

AI-229: Accessibility rules and generics

Add !work item status (as it was corrected after the ARG approved it).

The summary is incorrect, it doesn't reflect the wording.

Remove the comma before the “or within” in the wording.

Kiyoshi note that the fourth line of the discussion should be corrected: “possible” → “possibly”.

Approved with editorial changes: 7-0-1.

AI-233: Inheritance of components of generic formal derived types

Randy explains the purpose of this AI: to fix a problem with the corrigendum, and to resolve a question from a petitioner to the ACAA.

A lot of discussion occurs. Some ARG members believe that it is not clear that 7.3.1 applies here. A compromise position is reached: add “and 7.3.1” to the 12.5.1(20) paragraph to make the intent crystal clear.

John notes that in the question, “natural” should be capitalized.

Length should be capitalized in ‘length (and also in the question.)

Approved with editorial changes: 6-0-2.

AI-237 Finalization of task attributes

Germany rejected this AI during the WG9 vote. Why? Erhard responded that the second question of the problem section (about termination) is not answered. He thinks that answering this question may need to be done with a separate AI. Atomicity is found in the AI but not that it is directly connected to the question. There appears to be a patch possible within this AI.

Additionally, Erhard objects to the impression that the implementation can delay the termination of attributes at will. Instead, the impression should be to encourage implementation to do it sooner.

It appeared that the meeting converged towards the notion that finalization occurs no sooner than after the termination but it is permitted to be delayed until the master of the instantiation is completed. In an implementation advice the implementation should be encouraged to do it as soon as possible.

Erhard will rewrite.

AI-238: What is the lower bound of Ada.Strings.Bounded.Slice?

Make it a binding interpretation. Add wording: “The bounds of the returned string are Low and High.”

Kiyoshi notes that the !standard reference has nothing to do with this AI; correct it.

Approved with editorial changes: 7-1-0.

AI-239: Controlling inherited default expressions

Randy and Tucker explain the issue and the behavior of existing compilers. It is agreed that we must have wording for this change in order to determine that it is correct and doesn't cause any other problems before we can approve this.

Tucker suggests deleting “of type T” from 3.9.2(18) as a fix. Some question whether that is correct (it seems to allow too many cases).

Bob Duff will try to word the needed change.

Approved intent: 6-0-2.

AI-240: Stream attributes for limited types in Annex E

Change wording of E.2.2(8) to say “user-defined available attribute”, as “available” is necessary to include visibility issues. Fix discussion to match.

Approved with editorial changes: 7-0-1.

AI-242: Surprise behavior of Update

The consensus is that the language is insufficiently broken for the significant change proposed. It is really the equivalence rule in B.3.1(50) that is causing trouble. Change it to: To_C (Str, Append_Nul => False).

Approved intent: 5-0-3.

Randy will fix.

AI-246: View conversions between arrays of a by-reference type

The term “visibly by-reference” is not defined.

Fix the wording to replace visibly by-reference: “have a private or visibly by-reference subcomponent.” → “tagged, private, limited, or volatile subcomponent.”

Fix the wording: “then the neither type” → “then neither type”.

Change the summary to say anything that “is by-reference or might turn out to be” (that is private).

Change first paragraph of the summary to say “value conversion”; add “limited” to the second paragraph.

Approved intent: 6-0-3.

Randy will fix.

AI-259: Can accesses to atomic objects be combined?

Assigned to Bob Duff.

AI-272: Pragma Atomic and slices

Assigned to Pascal Leroy.

AI-273: Use of PCS should not be normative

Tucker suggested making the PCS optional, but that a substantially equivalent one is required. (That is, it could require the use of additional operations or additional parameters, but it should be similar where possible.)

Assigned to Gary Dismukes.