****
J3/02-296r3

Date: November 2002
To: J3
From: Aleksandar Donev
Subject: Reinstating Deferred Bindings
Reference: J3-007R3, J3/02-271r1
****

Summary

I propose to do the work necessary to bring back deferred bindings into F2x, and the specific syntax arrived at during a discussion with Kurt and Van.

J3 response:

J3 will recommend that deferred bindings be added to the draft standard, using the proposed specifications as a design guide.

Motivation

We want three useful debugging/diagnostic tools for deferred bindings:
1. They must be part of the syntax so that from the declaration of an extensible type it is clear that a particular type-bound procedure is deferred.
2. They must provide compile-time checks when extending the type, forcing the user to do something with the deferred procedure (either defer it again or override)
3. They should provide a runtime debugging protection/traceback if the deferred procedure is envoked.

Solution

We allow the following two forms of deferred binding:
a)
PROCEDURE, DEFERRED :: proc=>my_error_handler
The deferred attribute here means that upon extension another deferral or explicit override is required. This is a compile time check to the programmer.
b)
PROCEDURE, DEFERRED :: proc=>abstract_interface
In this case the compiler will automatically provide a *real* procedure with the specified interface which will do some kind of error-handling, and we say that invocation of this procedure will terminate execution (like invalid IO statements). This is a runtime aid to the programmer.

With this scheme deferred type-bound procedures are treated exactly like regular type-bound procedures, so the standard is simpler and there is no runtime segfaults. The same syntax applies to type-bound generics. There some of the specifics can be explicitly provided by the user, and others can be default debugging ones provided by the compiler.

Edits

Question:
1. How do we ensure that all DEFERRED bindings are overriden at compile time? The old text does not seem to have a numbered constraint for this.

44: 20-21 Replace:
If =><binding> does not appear and the binding does not override an inherited deferred binding,

45: 3+ Add:
or DEFERRED

45: 18+ Add:
or <deferred-procedure-name>

<deferred-procedure-name> is <name>

45: 10+ Add:
(Cxxx) If DEFERRED appears, either =><binding> shall appear, or the binding shall override an inherited binding.
DEFERRED and NON_OVERRIDABLE shall not both appear in the same <binding-attr-list>.

20+: Add:
(Cxxx) The <deferred-procedure-name> shall be the name of an abstract interface.
(Cxxx) The <deferred-procedure-name> shall appear only if a DEFERRED binding attribute appears and the binding does not override an inherited binding.

49: 4 Add to the end of the sentence:
or <deferred-procedure-name>.

49:5+ Add:
A binding that specifies the DEFERRED attribute is said to be deferred. If a deferred type-bound procedure is invoked execution is terminated.

[CONSTRAINT?] If a type has a deferred binding then any extension of the type shall override (4.5.3.2) this inherited deferred binding. This new binding may confirm that the binding is still deferred. It is permissible to override an inherited binding with a deferred binding.

NOTE

A deferred binding may be to a procedure or to an abstract interface. Deferred bindings are useful for type-bound procedures that do not have an actual implementation (as in Abstract Programming Interfaces). The user may provide a specific procedure in the binding that may give some useful run-time indication of the deferred character of the binding. But this deferred binding must still be explicitly overriden upon extension of the given type, which is a useful compile time reminder. The user may also just provide an abstract interface for the binding, in which case the processor may supply an error-handling procedure for the binding. This procedure terminates execution, but may also provide useful debugging diagnostics to the user.

56: 1- Add:
If a binding is deferred and does not specify an abstract interface then it inherits the interface from the binding in the parent type.