Subject: Updating complex parts
From: Van Snyder
Reference: 03-258r1, section 2.2.5, 04-225

# 1 Number

TBD

# 2 Title

Updating complex parts

# 3 Submitted By

J3

# 4 Status

For consideration.

# 5 Basic Functionality

Provide a syntax that allows to update the real and imaginary parts of a complex variable without updating the whole thing.

# 6 Rationale

It's not unusual to need to do this.

# 7 Estimated Impact

Very minor. Estimated at meeting 169 to be 4 on the JKR scale.

# 8 Detailed Specification

There are at least two ways to do this. One is to use a syntax similar to function reference, but allowed in variable-definition contexts. Another is to use a syntax similar to component reference, both in value-reference and variable-definition contexts.

## 8.1 Function-like syntax

Define a new variety of intrinsic procedure, the **accessor**. This is a procedure that can produce a value when invoked in a value-reference context, or can "absorb" a value and update (some or all of) its argument(s) when invoked in a variable-definition context. The argument(s) that is (are) updated have INTENT(OUT) or INTENT(INOUT), so the actual arguments can't be expressions, and can't be prohibited to appear in variable-definition contexts.

The following intrinsic procedures would be useful accessors. Their behavior in the case when they are invoked in a variable-definition context is described here. Their behavior when invoked in a value-reference context would not be affected. The equivalent behavior shown below could frequently result in construction of an array temp, so this proposal might have some performance benefits.

Accessors cannot be actual arguments because that would essentially entail providing for user-defined accessors.

| Name | Functionality |
|---|---|
| ABS | Update the magnitude of a variable. For a noncomplex variable, `abs(x) = y` is equivalent to `x = sign(y,x)`. For a complex variable, `abs(x) = y` is equivalent to `temp = atan2(aimag(x),real(x)); x = abs(y) * cmplx(cos(temp),sin(temp))`. This is mathematically equivalent to `abs(y) * x / abs(x)`, but it is necessary to take care of the case that `x == 0.0`. |
| AIMAG | Update the imaginary part of a variable. `aimag(x) = y` is equivalent to, but probably cheaper than `x = cmplx(real(x),y))`. |
| EXPONENT | Update the exponent part of a floating-point variable. `exponent(x) = j` is equivalent to `x = set_exponent ( fraction(x), j)`. |
| FRACTION | Update the fraction part of a floating-point variable. `fraction(x) = y` is equivalent to `x = set_exponent ( y, exponent(x) )`. |
| IBITS | Update bits POS through POS + LEN − 1 of I. `ibits(i,pos,len) = j` is equivalent to `call mvbits ( j, 0, len, i, pos )`. `call mvbits ( j, frompos, len, i, topos )` is equivalent to `ibits ( i, topos, len ) = ibits ( j, frompos, len )`. |
| MERGE | Update TSOURCE or FSOURCE depending on MASK. `merge(x,y,m) = z` is equivalent to `where ( m ); x = z; elsewhere; y = z; endwhere`, or an equivalent IF construct if `x`, `y`, and `m` are scalars. |
| REAL | Update the real part of a complex variable. The KIND argument is not permitted (because it would be nonsense). `real(x) = y` is equivalent to, but probably cheaper than `x = cmplx(y,aimag(x))`. |

It is conceivable that updating behavior could be defined for PACK and UNPACK.

## 8.2 Component-like syntax

Specify that the real and imaginary parts of a complex variable can be accessed by using component-like syntax, with "component" names REAL and AIMAG. It might be possible simply to define COMPLEX to be a SEQUENCE derived type with components named, say, REAL and AIMAG, in that order, because C424 prohibits other definitions of COMPLEX, with or without the same components in the same order. A user-defined type therefore could not be "equivalent" to COMPLEX but without the requisite behavior.

It is conceivable that the rest of the above, except for IBITS and MERGE, could be done using component-like syntax, but they could not be done simply by defining an intrinsic type to be a sequence derived type. The function-like syntax is more powerful, and has an obvious generalization to user-defined procedures.

# 9 History