Generic Arrays

An Alternative to the Name Brands

Jim Leek, Tom Epperly, & Gary Kumfert

Center for Applied Scientific Computing

January 27, 2005

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.
UCRL-PRES-209185

A Generic What-sis?

- A generic array is an array super type.
  - Any array may be passed as a generic array
  - A generic array has no static dimension or type
  - A generic array's data cannot be accessed
  - A generic array's meta-data may be accessed, dimension, upper, lower, type, order, etc.
  - A generic array may be 'cast' to a type and dimension.

Who needs it?

- Anyone who needs a method to be able to take multiple different kinds of arrays.
  - Generic arrays were originally requested by the Scientific Data Components and Interfaces Working Group.

Using a Generic Array

- A Generic array may be
  - AddRef'd DeleteRef'd
  - smartcopy'd
- Data may be accessed after the generic array is cast to a basic type.
  - bool, int, dcomplex, etc. OK
  - object array. NO
- The type() function returns type as an enumeration.

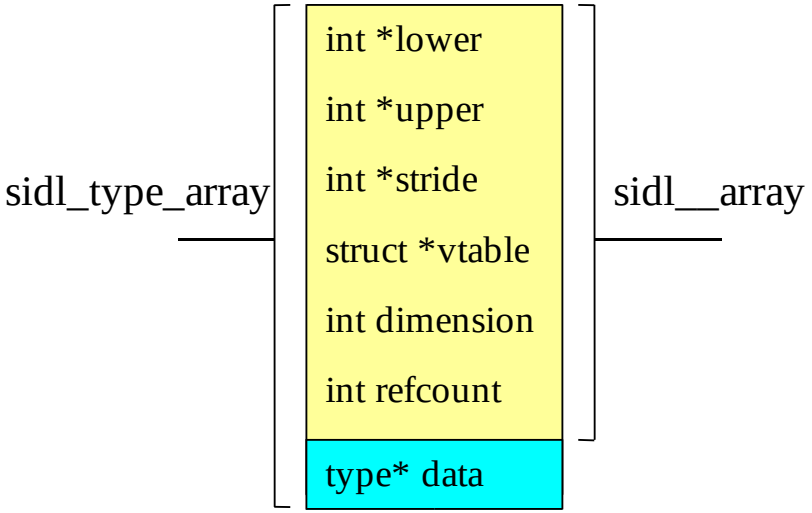
SIDL Syntax

- Generic arrays are a type, like int, or bool.
  - Example:
    - array<> passGeneric(in array<> inArg,
      inout array<> inOutArg,
      out array<> outArg);

Generic Arrays in C (IOR)

- The generic array type name is “struct sidl__array”
- Functions are called in the usual manner:
  - int32_t sidl__array_type(struct sidl__array* array

- Casting is done with a standard C cast.
- The generic array type holds all the meta-data for any array.



C example code

```
struct sidl__array* impl_ArrayTest_ArrayOps_passGeneric(
  /*in*/ struct sidl__array* inArg) {
/* DO-NOT-DELETE splicer.begin(ArrayTest.ArrayOps.passGeneric) */
  int32_t lower[7], upper[7], i;
  struct sidl__array *result = NULL;
  if (sidl__array_type(inArg) == sidl_bool_array) {
   struct sidl_bool__array * a = (struct sidl_bool__array *)inArg;
      /* DO SOMETHING…….*/
   }
return result;
 /* DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric) */
```

Generic Arrays in Cxx and UCxx

- In C++ the generic array is literally a super class of other arrays called sidl::basearray
- The type function in C++ is arrayType().
- A standard static_cast can be used to cast the generic array.

Cxx/UCxx Example

```
::sidl::basearray ArrayTest::ArrayOps_impl::passGeneric (
/*in*/ ::sidl::basearray inArg ) throw () {
// DO-NOT-DELETE splicer.begin(ArrayTest.ArrayOps.passGeneric)
  int32_t lower[7], upper[7], i;
  ::sidl::basearray result = NULL;
  if (inArg.arrayType() == sidl_bool_array) {
     ::sidl::array<bool > & temp =
        static_cast< ::sidl::array< char > &>(inArg);
     // DO SOME STUFF…..
}
return result;
 // DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric)
```

Generic Arrays in Fortran 77

- As usual, Generic Arrays are passed in as a 64-bit integer.
- Methods are called by the usual method (these may be called on any array)
  - call sidl__array_type(a)
- There is no need to cast.

Fortan 77 Example

```
 subroutine ArrayTest_ArrayOps_passGeneric_fi(inArg, retval)
       implicit none
C       in array<> inArg
       integer*8 inArg
C       out array<> retval
       integer*8 retval
C     DO-NOT-DELETE splicer.begin(ArrayTest.ArrayOps.passGeneric)
C     Insert the implementation here...
      integer*4 lower(7), upper(7), i, type, dimen, outdimen
      call sidl__array_type_f(inArg, type)
      if (type .eq. 1) then
C      DO SOME STUFF
       end if
C     DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric)
       end
```

Generic Arrays in FORTRAN 90

- Generic arrays in Fortran 90 have the type sidl__array.
- The generic array module is called sidl_array_array
- The module includes the standard generic array functions is short form:
  - tp = type(a); call smartcopy(src,dest)
- Casting is done by this cast function:
  - cast(<source>, <target>)

Fortran 90 Example

```
recursive subroutine ArrayO_passGenericwdjmxoh8x1_mi(inArg,  retval)
  use ArrayTest_ArrayOps
  use sidl_array_array
  use ArrayTest_ArrayOps_impl
  ! DO-NOT-DELETE splicer.begin(ArrayTest.ArrayOps.passGeneric.use)
  use sidl_int_array
  ! DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric.use)
  implicit none
  type(sidl__array) :: inArg ! in
  type(sidl__array) :: retval ! out
  ! DO-NOT-DELETE splicer.begin(ArrayTest.ArrayOps.passGeneric)
   type(sidl_bool_1d) :: b1
  integer (selected_int_kind(9)) :: lw(7), up(7), i, tp, dmn, outdimen
  if (type(inArg) .eq. 1) then
     call set_null(b1)
     call cast(inArg, b1)
  end if
! DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric)
end subroutine ArrayO_passGenericwdjmxoh8x1_mi
```

Generic Arrays in Java

- In Java, the Generic Array type is a super type of all arrays: gov.llnl.sidl.BaseArray
- Casting is done with a simple Java cast.
- A _dcast() call is required to resolve the dimension.
- The type function may be used.
- class BaseArray includes all the standard functions

Java Example

```
public static gov.llnl.sidl.BaseArray passGeneric_Impl (
    /*in*/ gov.llnl.sidl.BaseArray inArg)  {
    // DO-NOT-DELETE splicer.begin
     (ArrayTest.ArrayOps.passGeneric)
    int lower[] = new int[7];
    int upper[] = new int[7];
    int i;
    gov.llnl.sidl.BaseArray result = null;
    if (inArg._type() == sidl_bool_array) {
     sidl.Boolean.Array ba = (sidl.Boolean.Array) inArg;
     sidl.Boolean.Array1 ba1 = (sidl.Boolean.Array1) ba._dcast();
      //DO SOMETHING
    }
    return result;
    // DO-NOT-DELETE splicer.end(ArrayTest.ArrayOps.passGeneric)
}
```

Generic Arrays in Python

- Python is very simple, since it is dynamically typed.
- Python is unlike any due to Numeric. There is no 'type' function, only typecode()

Python Example

```
def passGeneric(inArg):
# DO-NOT-DELETE splicer.begin(passGeneric)
  ret = None
  if (inArg.typecode() == 'c'): #char array
    #DO SOME STUFF
  return (ret)
# DO-NOT-DELETE splicer.end(passGeneric)
```

Conclusion

- Generic Arrays are a simple way to generalize your SIDL interface.
- Generic Arrays are fairly natural in every language, although the interface varies.
- We hope you will find Generic Arrays a useful addition to the Babel toolkit.