Definingan

Postgres supports array types in a fashion similar to Pascal - that is, the array type must be a separate type rather than multiple values of an existing type. Unlike Pascal, however, Postgres array types can be both fixed or variable length.

Just as for any other Postgres type, array types must have input and output functions defined for them. In the future, Postgres may be able to use the array element in/out functions for input and output, but currently you will have to write separate ones for arrays. See the tutorial "Defining User Defined Types, Operators, and Functions" for more details on this. Operators can be defined for array types in the same manner as they are for any other types.

For the purpose of this tutorial, suppose we want to have an array of four longs (and that sizeof(long) is 4) to be called "quarterly". Also, suppose we will have a variable length array of longs called "stats".

The only special things we need to know when writing the in/out functions for "quarterly" is that Postgres will pass a "simple" array of longs to the output function and expect a "simple" array of longs in return from the input function. A sample "simple" array suitable for quarterly can be declared as follows:

```
long quarterly[4];
```

For the variable length array "stats", the situation is a little more complicated. Because Postgres will not know in advance how big the array is, Postgres will expect the length of the array (in bytes) to be encoded in the first four bytes of the memory which contains the array. The expected structure is as follows:

```
typedef struct
{
   long length;
   unsigned char bytes[1]; /* Force contiguity */
}
VAR_LEN_ATTR;
```

The in function for the stats array will look something like this:

```
VAR_LEN_ATTR *stats_in(s)

char s;

{
   VAR_LEN_ATTR *stats;
   long array_size, *arrayp;

   ... figure out how long the stats array should be ...

   /* We need the extra sizeof(long) to hold the length */

   stats = (VAR_LEN_ATTR *) palloc(array_size * sizeof(long) + sizeof(long));

   /* length is length of the entire structure in bytes, INCLUDING ITSELF */

   stats->length = array_size * sizeof(long) + sizeof(long);

   arrayp = &(stats->bytes[0]);

   for (i = 0; i < length; i++, arrayp++)
   {
      ... populate the stats array with numbers from s ....
   }
```

```
    return(stats);
}
```

The output function for stats will get the same VAR_LEN_ATTR structure.

Now, using the Define C Function command (assuming the functions are in /usr/postgres/tutorial/stats.c and /usr/postgres/tutorial/quarterly.c), we can define our two arrays. First we will define the fixed-size array "quarterly". Note that internallength is known to be 16, because

```
sizeof(long) * 4 = 4 * 4 = 16
```

on our hypothetical machine.

First we define the in/out functions:

```
define c function quarterly_in (file = "/usr/postgres/tutorial/quarterly.c",
    returntype = quarterly) arg is (char16)

define c function quarterly_out (file = "/usr/postgres/tutorial/quarterly.c",
    returntype = char16) arg is (quarterly)
```

And now we can define the type:

```
define type quarterly (internallength = 16,
            input = quarterly_in, output = quarterly_out,
            element = int4)
```

Now we define the stats array. First we define its in/out functions:

```
define c function stats_in (file = "/usr/postgres/tutorial/stats.c",
    returntype = stats) arg is (char16)

define c function stats_out (file = "/usr/postgres/tutorial/stats.c",
    returntype = char16) arg is (stats)
```

and now we define the type. Note that setting internallength to 0 tells Postgres that it is a variable length type.

```
define type stats (internallength = 0,
          input = stats_in, output = stats_out,
          element = int4)
```

Now we can run some queries:

```
create test (a = quarterly, b = stats)

append test (a = "1 2 3 4"::quarterly, b = "5 6 7"::stats)
append test (a = "1 3 2 4"::quarterly, b = "6 4"::stats)

retrieve (test.all) where test.a[3] = test.b[1]

------------------
|    a    |  b  |
-------------------
| 1 3 2 4  | 6 4 |
------------------
```