Subject: Re: Problems with IDL call_external to C shared object
Posted by Mark Rivers on Thu, 12 Oct 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I've done a lot of this, and it's not that hard, don't give up!

Here are some tricks:

- All arrays which need to be passed between IDL and C must be allocated in IDL, as J.D. Smith said. This includes both arrays being passed from IDL to C and from C back to IDL. Sometimes this requires an initial call to the C code to return the array sizes which IDL will allocate, if the array sizes are not known to IDL beforehand.

- Don't deallocate any arrays which were passed from IDL.

- Don't pass strings, rather pass byte arrays. It is much simpler. Convert strings to byte arrays in IDL before or after the CALL_EXTERNAL call.

- Convert all output variables to the data type which C is expecting in the CALL_EXTERNAL call.

> - What is the effect of the /CDECL keyword to CALL_EXTERNAL ?
> I tried with and without but no success.

This controls the calling convention. If your C function is being called then you probably have this set correctly.

> - Is it possible that the C program "forgets" something between
> the IDL CALL_EXTERNALs ?

- As J.D. Smith said, it will forget anything which is not global or static.

> - How can I return an array via CALL_EXTERNAL or have I always
> to loop over calls returning scalars ? The EZCA library (channel
> access to EPICS control system) manages to return arrays, but I
> couldn't figure out how.

My EZCA code is rather opaque, since it uses macros which allow it to work on both IDL and PV-WAVE, on Unix, VMS and Windows.

Here is a simple example. It is C code which computes the Mandelbrot set, and is called from IDL.
argv[7] is a 2-D array.

```
void mandelbrot(int argc, void *argv[])
{
 int nr = *(int *) argv[0];
```

```
 int ni = *(int *) argv[1];
 double rstart = *(double *) argv[2];
 double istart = *(double *) argv[3];
 double dr = *(double *) argv[4];
 double di = *(double *) argv[5];
 int max_iter = *(int *) argv[6];
 int *result =  argv[7];
int i, j, count;
 double real, imag, rz, iz, sz2, rz2, iz2;
    for (i=0; i<ni; i++) {
      imag = istart + i*di;
      for (j=0; j<nr; j++) {
           real = rstart + j*dr;
           rz = 0.;
           iz = 0.;
           sz2 = 0.;

           count = 0;
           while ((count < max_iter) && (sz2 < 4.0)) {
                rz2 = rz * rz;
                iz2 = iz * iz;
                iz = 2.0 * rz * iz + imag;
                rz = rz2 - iz2 + real;
                sz2 = rz2 + iz2;
                count++;
           }
           *result++ = count;
      }
    }
}
```

Here is the IDL code which calls the C code:

```
function mandelbrot1, xcenter, ycenter, radius, size, max_iter, xout, yout
if (n_elements(size) eq 0) then size=100
if (n_elements(max_iter) eq 0) then max_iter=255
dx = double(radius)*2/size
xstart = double(xcenter - radius)
xstop = double(xcenter + radius)
ystart = double(ycenter - radius)
ystop = double(ycenter + radius)
result = lonarr(size, size)
xout = xstart + findgen(size)*dx
yout = ystart + findgen(size)*dx
s = call_external('mandelbrot.dll', 'mandelbrot', $
                  long(size), $
                  long(size), $
```

```
                double(xstart), $
                double(ystart), $
                double(dx), $
                double(dx), $
                long(max_iter), $
                result)
return, result
end
```