Subject: Re: Win32 jacket routines between IDL and third-party DLL
Posted by rivers on Wed, 06 Jan 1999 08:00:00 GMT
View Forum Message <> Reply to Message

In article <76vl1k$h41$1@willow.cc.kcl.ac.uk>, nra@maxwell.ph.kcl.ac.uk (Nigel Arnot) writes:
> I have very little programming experience in the Win32 environment, so
> apologies if this is a trivial question.
>
> The problem I'm looking at is that a colleague wants to use an IDL front-end
> program to control a CAMAC crate. The crate controller comes with a set of
> C-callable routines in the form of a DLL, but the arguments expected aren't
> in the form that IDL's CALL_EXTERNAL will generate.
>
> So, what's needed are jacket routines that accept arguments from IDL, and then
> generate calls to the CAMAC controller routines.
>
> Is it straightforward to write a DLL called from IDL that in turn calls
> another DLL, and where do I look up the details?

Yes, this is easy. You are effectively doing this whenever your C programs call any of the C runtime library routines, or call the WIN32 API routines.

> If anyone has done something like this already, I'd much appreciate a peek at
> the code!

I have done lots of coding like that. I control CAMAC crates from IDL under VMS, and I am about to do it under Windows NT, using the Kinetic Systems crate controller and their DLL.

The only code I have right now looks quite complex, because I have written it with macros which allow it to work under both IDL and PV-WAVE, and under VMS, Unix, and Windows. All of these systems pass their arguments (especially strings) differently.

Here is the beginning of that file:

```
#include <tsDefs.h>
#include <cadef.h>
#include <ezca.h>

#include <Ezca.h>
#define OK      0
#undef ERROR
#define ERROR   -1
extern struct caGlobals CA;

/*  ezcaIDL.c
```

```
*
* This file is an interface layer between IDL/PV-WAVE and EZCA.  It mainly
*   just converts parameter passing from the IDL/PV-WAVE mechanisms
*   (most parameters passed by reference) to the mechanism required by EZCA.
*   For a few the functions which are not supplied by EZCA this routine actually
*   does the work, using the ezcaPvToChid to fetch the channel descriptor.
*
* Author: Mark Rivers
* Date: June 28, 1995
*/

/* The following macros allow for the differences in the way PV-WAVE and
*  IDL pass character strings. PV-WAVE passes the address of the address
*  of the string in argp[]. IDL passes the address of a string descriptor
*  structure, which contains the address of the string as one of its elements */
#ifdef IDL
typedef struct {
   unsigned short length;
   short type;
   char *address;
} IDL_STR_DESCR;
#define STRARG IDL_STR_DESCR
#define STRADDR(s) s->address
#define STRFIXLEN(s) s->length = strlen(s->address)
static char* str_array_addr[1000];
#define BUILD_STR_ARRAY(len, s) for (i=0; i<len; i++) str_array_addr[i]=s[i].address
#define STR_ARRAY_ADDR(s) str_array_addr

#else /*(PV-WAVE)*/
#define STRARG char*
#define STRADDR(s) *s
#define STRFIXLEN(s) (*s)[strlen(*s)] = (char) 32
#define BUILD_STR_ARRAY(len, s)
#define STR_ARRAY_ADDR(s) s
#endif

/* The following macros allow this source file to be used with
   PV-WAVE and IDL on both Unix and VMS platforms. The difference
   is that on VMS platforms arguments are passed directly
   (by reference), while on Unix they are passed by the (argc, argp)
   mechanism. These macros also simplify the code for each routine. */
#if defined (VMS)
#   define WAVE_HEADER0(ftype, fname)\
      ftype fname() {
#   define WAVE_HEADER1(ftype, fname, type1, arg1)\
      ftype fname(type1 *arg1) {
#   define WAVE_HEADER2(ftype, fname, type1, arg1, type2, arg2)\
      ftype fname(type1 *arg1, type2 *arg2) {
```

```
#   define WAVE_HEADER3(ftype, fname, type1, arg1, type2, arg2, type3, arg3)\
     ftype fname(type1 *arg1, type2 *arg2, type3 *arg3) {
#   define WAVE_HEADER4(ftype, fname, type1, arg1, type2, arg2, type3, arg3, type4, arg4)\
     ftype fname(type1 *arg1, type2 *arg2, type3 *arg3, type4 *arg4) {
#   define WAVE_HEADER5(ftype, fname, type1, arg1, type2, arg2, type3, arg3, type4, arg4,
type5, arg5)\
     ftype fname(type1 *arg1, type2 *arg2, type3 *arg3, type4 *arg4, type5 *arg5) {

#else
#   define WAVE_HEADER0(ftype, fname)\
   ftype fname(argc, argp)\
    int argc;\
    void *argp[];\
    {
#   define WAVE_HEADER1(ftype, fname, type1, arg1)\
   ftype fname(argc, argp)\
    int argc;\
    void *argp[];\
    {\
    type1 *arg1 = (type1 *) argp[0];
#   define WAVE_HEADER2(ftype, fname, type1, arg1, type2, arg2)\
   ftype fname(argc, argp)\
    int argc;\
    void *argp[];\
    {\
    type1 *arg1 = (type1 *) argp[0];\
    type2 *arg2 = (type2 *) argp[1];
#   define WAVE_HEADER3(ftype, fname, type1, arg1, type2, arg2, type3, arg3)\
   ftype fname(argc, argp)\
    int argc;\
    void *argp[];\
    {\
    type1 *arg1 = (type1 *) argp[0];\
    type2 *arg2 = (type2 *) argp[1];\
    type3 *arg3 = (type3 *) argp[2];
#   define WAVE_HEADER4(ftype, fname, type1, arg1, type2, arg2, type3, arg3, type4, arg4)\
   ftype fname(argc, argp)\
    int argc;\
    void *argp[];\
    {\
    type1 *arg1 = (type1 *) argp[0];\
    type2 *arg2 = (type2 *) argp[1];\
    type3 *arg3 = (type3 *) argp[2];\
    type4 *arg4 = (type4 *) argp[3];
#   define WAVE_HEADER5(ftype, fname, type1, arg1, type2, arg2, type3, arg3, type4, arg4,
type5, arg5)\
   ftype fname(argc, argp)\
    int argc;\
```

```
    void *argp[];\
    {\
    type1 *arg1 = (type1 *) argp[0];\
    type2 *arg2 = (type2 *) argp[1];\
    type3 *arg3 = (type3 *) argp[2];\
    type4 *arg4 = (type4 *) argp[3];\
    type5 *arg5 = (type5 *) argp[4];
#endif


WAVE_HEADER4(int, ezcaIDLGet, STRARG, pvname, char, type, int, nelem, void, buff)
   return(ezcaGet(STRADDR(pvname), *type, *nelem, buff));
}

WAVE_HEADER3(int, ezcaIDLGetControlLimits, STRARG, pvname, double, low, double, high)
   return(ezcaGetControlLimits(STRADDR(pvname), low, high));
}

WAVE_HEADER3(int, ezcaIDLGetGraphicLimits, STRARG, pvname, double, low, double, high)
   return(ezcaGetGraphicLimits(STRADDR(pvname), low, high));
}

WAVE_HEADER2(int, ezcaIDLGetPrecision, STRARG, pvname, short, precision)
   return(ezcaGetPrecision(STRADDR(pvname), precision));
}

WAVE_HEADER4(int, ezcaIDLGetStatus, STRARG, pvname, TS_STAMP, timestamp,
short, status, short, severity)
   return(ezcaGetStatus(STRADDR(pvname), timestamp, status, severity));
}

WAVE_HEADER2(int, ezcaIDLGetUnits, STRARG, pvname, char, units)
   return(ezcaGetUnits(STRADDR(pvname), units));
}

WAVE_HEADER1(int, ezcaIDLPerror, char, prefix)
   ezcaPerror(prefix);
   return EZCA_OK;
}

WAVE_HEADER2(int, ezcaIDLGetErrorString, char, prefix, char, err_string)
 int status;
 char *buff;

 status = ezcaGetErrorString(prefix, &buff);
 strcpy(err_string, buff);
 ezcaFree((void *)buff);
 return(status);
```

```
}

WAVE_HEADER4(int, ezcaIDLPut, STRARG, pvname, char, type, int, nelem, void, buff)
  return(ezcaPut(STRADDR(pvname), *type, *nelem, buff));
}

WAVE_HEADER3(int, ezcaIDLGetCountAndType, STRARG, pvname, int, count, char, type)
  int status;
  chid *chid;

  status = ezcaPvToChid(STRADDR(pvname), &chid);
  if (status != EZCA_OK) return(status);
  *count = ca_element_count(*chid);
  *type = ca_field_type(*chid);
  return EZCA_OK;
}

WAVE_HEADER2(int, ezcaIDLSetMonitor, STRARG, pvname, char, type)
  return(ezcaSetMonitor(STRADDR(pvname), *type));
}

WAVE_HEADER2(int, ezcaIDLClearMonitor, STRARG, pvname, char, type)
  return(ezcaClearMonitor(STRADDR(pvname), *type));
}

WAVE_HEADER2(int, ezcaIDLNewMonitorValue, STRARG, pvname, char, type)
  return(ezcaNewMonitorValue(STRADDR(pvname), *type));
}

WAVE_HEADER1(int, ezcaIDLSetTimeout, float, sec)
  return(ezcaSetTimeout(*sec));
}

WAVE_HEADER1(int, ezcaIDLSetRetryCount, int, retry)
  return(ezcaSetRetryCount(*retry));
}

WAVE_HEADER1(int, ezcaIDLGetTimeout, float, timeout)
  *timeout = ezcaGetTimeout();
  return EZCA_OK;
}

WAVE_HEADER1(int, ezcaIDLGetRetryCount, int, retrycount)
  *retrycount = ezcaGetRetryCount();
  return EZCA_OK;
}

WAVE_HEADER2(int, ezcaIDLEndGroupWithReport, int, nvals, int, status)
```

```
   int rc, nrcs;
   int *rcs;
   int i;

   rc = ezcaEndGroupWithReport(&rcs, &nrcs);
   if (nrcs > *nvals) nrcs = *nvals;
   if (*nvals > nrcs) *nvals = nrcs;
   for (i=0; i<*nvals; i++) status[i] = rcs[i];
   ezcaFree((void *) rcs);
   return(rc);
}
```