Subject: Re: Dynamic creation of PV-Wave variables in C
Posted by rivers on Wed, 24 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <4t3fj7$m5e@mksrv1.dseg.ti.com>, JH7@msg.ti.com (John Houston) writes:
> I have some old C code which is designed to read data out of a binary
> file. The length and type of data is encoded into the data file
> itself. I would like to know if it is possible to use my C routines
> to read the data and then transfer the data to PV-Wave. The data is
> dynamic in length and structure. Any example code would be
> appreciated.

There are at least 2 ways to do this.

1) Using CALL_EXTERNAL
Make one call to your C routines via CALL_EXTERNAL to determine the length and type of your data. In IDL allocate the arrays to hold this data, and then call your C routines again via CALL_EXTERNAL to actually read the data, passing the pointers to the IDL array variables.

2) Using LINK_IMAGE
In this case a single call to a specially written C routine will suffice. This C routine can crate new IDL variables of the correct type and size. I am appending an example of this. This example is a C "glue" routine between IDL and my C routines which actually read the header and data from the file. My C routines contain the functions like "open_image_file", "read_image_header", "read_image_record", etc.

```
#include <stdio.h>
#include "idl_dir:[external]export.h"
#include "idl_dir:[external]obsolete.h"
#include "sys_imaging:image_header.h"

read_bsif_i(argc, argv, argk)
  int argc;
  VPTR argv[];
  char *argk;

  {
#define FILE_DS     argv[0]
#define N_ROWS_DS   argv[1]
#define N_COLS_DS   argv[2]
#define N_DATA_DS   argv[3]
#define X_NORMAL_DS argv[4]
#define Y_NORMAL_DS argv[5]
#define ROTATED_DS  argv[6]
#define X_START_DS  argv[7]
#define X_STOP_DS   argv[8]
```

```
#define Y_START_DS  argv[9]
#define Y_STOP_DS   argv[10]
#define IMAGE_TITLE_DS argv[11]
#define X_TITLE_DS  argv[12]
#define Y_TITLE_DS  argv[13]
#define DATA_TITLE_DS argv[14]
#define DATA_TYPE_DS argv[15]
#define COMP_TYPE_DS argv[16]
#define USER_BUFFER_DS argv[17]
#define IMAGE_DATA_DS argv[18]

/*  This routine reads BSIF files into IDL. It is quite general.
    The only restrictions are:
       Images with multiple data values per pixel must all be the same
       data type. It reads all of the data into a single array =
                image_data(n_cols, n_rows, n_data)
       The compresion type, data_type, min and max of each datum are not
       returned. These are either not needed or are easily determined within
       IDL itself.
*/

#define NEW_FILE FALSE
#define MAX_DATA 32
#define MAX_TITLE_LEN 132
#define MAX_USER_BUFFER_LEN 8192

      void close_image_file();
      int open_image_file(), read_image_record(), read_image_header();
      int row, datum, i;
      int n_rows, n_cols, n_data;
      int file_id;
      int dim_blk[3];
      int *address, data_size, type;
      int temp_int;
      int user_buffer_len;
      char image_title[MAX_TITLE_LEN];
      char x_title[MAX_TITLE_LEN];
      char y_title[MAX_TITLE_LEN];
      char data_title[MAX_DATA][MAX_TITLE_LEN];
      int image_title_len, x_title_len, y_title_len;
      int data_title_len[MAX_DATA];

      UCHAR user_buffer[MAX_USER_BUFFER_LEN];
      UCHAR *base_address;

    struct image_descr image_descr;
    struct data_descr data_descr[MAX_DATA];
    STRING scratch_string;
```

```
    STRING *string_ptr;
    VPTR   temp;
    char buf[128];


/*     Check parameters */
      if (argc != 19) {
        message(M_NAMED_GENERIC, 0, MSG_LONGJMP,
              "Incorrect number of parameters");
        return;
       }

      temp_int = FILE_DS->value.str.slen;
      if (open_image_file(&file_id,
          FILE_DS->value.str.s,
          &temp_int,
          &FALSE) != 1) {
            sprintf(buf, "Error opening file %s", FILE_DS->value.str.s);
            message(M_NAMED_GENERIC, 0, MSG_LONGJMP, buf);
        return;
       }

      temp_int = MAX_USER_BUFFER_LEN * 4;
      if (read_image_header(&file_id, &MAX_DATA, &n_data,
              &image_descr, &data_descr, &MAX_TITLE_LEN,
              &image_title, &image_title_len,
              &x_title, &x_title_len,
              &y_title, &y_title_len,
              &data_title, &data_title_len,
              &user_buffer, &temp_int,
              &user_buffer_len) != 1) {
        message(M_NAMED_GENERIC, 0, MSG_LONGJMP,
               "Error reading image header");
        return;
      }

/*     Copy a few values for shorthand */
      n_cols = image_descr.n_cols;
      n_rows = image_descr.n_rows;
      n_data = image_descr.n_data;

/*    Copy the scalar parameters from the image header */
      store_scalar(N_ROWS_DS, TYP_LONG,    (ALLTYPES *)&image_descr.n_rows);
      store_scalar(N_COLS_DS, TYP_LONG,    (ALLTYPES *)&image_descr.n_cols);
      store_scalar(N_DATA_DS, TYP_LONG,    (ALLTYPES *)&image_descr.n_data);
      store_scalar(X_NORMAL_DS, TYP_LONG,    (ALLTYPES *)&image_descr.x_normal);
      store_scalar(Y_NORMAL_DS, TYP_LONG,    (ALLTYPES *)&image_descr.y_normal);
      store_scalar(ROTATED_DS, TYP_LONG,    (ALLTYPES *)&image_descr.rotated);
```

```
store_scalar(X_START_DS, TYP_DOUBLE, (ALLTYPES *)&image_descr.x_start);
store_scalar(X_STOP_DS, TYP_DOUBLE, (ALLTYPES *)&image_descr.x_stop);
store_scalar(Y_START_DS, TYP_DOUBLE, (ALLTYPES *)&image_descr.y_start);
store_scalar(Y_STOP_DS, TYP_DOUBLE, (ALLTYPES *)&image_descr.y_stop);

/* Copy the string parameters */
str_store(&scratch_string, (char *)&image_title);
store_scalar(IMAGE_TITLE_DS, TYP_STRING, (ALLTYPES *)&scratch_string);
str_store(&scratch_string, (char *)&x_title);
store_scalar(X_TITLE_DS,     TYP_STRING, (ALLTYPES *)&scratch_string);
str_store(&scratch_string, (char *)&y_title);
store_scalar(Y_TITLE_DS,     TYP_STRING, (ALLTYPES *)&scratch_string);


 /* Copy the data titles */
make_temp_array(TYP_STRING, 1, (LONG*)&n_data, BASICARR_INI_NOP, &temp);
var_copy(temp, DATA_TITLE_DS);
string_ptr = (STRING *)DATA_TITLE_DS->value.arr->data;
for(i=0; i<n_data; i++) str_store(&string_ptr[i], &data_title[i][0]);

/* Copy the data types */
make_temp_array(TYP_LONG, 1, (LONG*)&n_data, BASICARR_INI_NOP, &temp);
var_copy(temp, DATA_TYPE_DS);
address = (int *)DATA_TYPE_DS->value.arr->data;
for (i=0; i<n_data; i++) *address++ = data_descr[i].data_type;

/* Copy the compression types */
make_temp_array(TYP_LONG, 1, (LONG*)&n_data, BASICARR_INI_NOP, &temp);
var_copy(temp, COMP_TYPE_DS);
address = (int *)COMP_TYPE_DS->value.arr->data;
for (i=0; i<n_data; i++) *address++ = data_descr[i].compression_type;

/* Copy the user buffer */
make_temp_array(TYP_BYTE, 1, (LONG*)&user_buffer_len, BASICARR_INI_NOP, &temp);
var_copy(temp, USER_BUFFER_DS);
base_address = USER_BUFFER_DS->value.arr->data;
for (i=0; i<user_buffer_len; i++) *base_address++ = user_buffer[i];

/* Create the image data array */
switch (data_descr[0].data_type) {

case UB_TYPE:
case SB_TYPE:
  type = TYP_BYTE;
  data_size = 1;
  break;
case US_TYPE:
case SS_TYPE:
```

```
      type = TYP_INT;
      data_size = 2;
      break;
    case UL_TYPE:
    case SL_TYPE:
      type = TYP_LONG;
      data_size = 4;
      break;
    case FL_TYPE:
      type = TYP_FLOAT;
      data_size = 4;
      break;
    case DL_TYPE:
      type = TYP_DOUBLE;
      data_size = 8;
      break;
    }

    dim_blk[0] = image_descr.n_cols;
    dim_blk[1] = image_descr.n_rows;
    dim_blk[2] = image_descr.n_data;

    make_temp_array(type, 3, (LONG*)&dim_blk, BASICARR_INI_NOP, &temp);

    var_copy(temp, IMAGE_DATA_DS);

    base_address = IMAGE_DATA_DS->value.arr->data;
    for (row=0; row<n_rows; row++) {
      for (datum=0; datum<n_data; datum++) {
        address = (int *) (base_address +
          ((datum*n_rows*n_cols) + row*n_cols) * data_size);
        if (read_image_record(&file_id, &image_descr,
                  &data_descr[datum], address) != 1) {
          sprintf(buf, "Error reading image data in row %d", row+1);
          message(M_NAMED_GENERIC, 0, MSG_LONGJMP, buf);
          return;
        }
      }
    }
    close_image_file(&file_id);
}
```