Subject: Returning a string array from a DLM
Posted by Ed Wright on Thu, 12 Dec 2002 23:44:45 GMT
View Forum Message <> Reply to Message

Question for DLM experts:

I wrote a DLM for a C routine that returns an array of strings. The first version of the DLM code has the return array declared as char item[X][Y]. After the C call, I created a temp array of strings via a IDL_MakeTempArray call, then looped over each string in the array copying the string to the array created in IDL_MakeTempArray via an IDL_StrStore call.

It worked..

So being quite industrious, I decided it a good idea to dynamically allocate the memory required for the string array. After a few hours of aggravation and a phone call to a friend who reminded me a pointer pointing to nothing might cause some problems, I have a DLM than can dynamically allocate the needed space for a string array then pass that array to my C routine as an argument for return.

Now the problem. On return to the IDL interpreter, I experience a plethora of malloc errors:

Given a script making two calls to the DLM, call format:

cspice_lparse, string_with_separator, separator_string, item

Item should return as an array of strings.

My test output -

IDL> xtspice_idl
Start tests
% Loaded DLM: COMBINE.
Start of module: st01_idl
In C routine:
 String: a
 String: b
 String: c
In IDL, array = a b c

In C routine:
 String: a
 String:
 String: c
*** malloc[3990]: error for object 0x29cee0: Incorrect check sum for freed object - object was probably modified after beeing freed; break at

szone_error
Segmentation fault

Any ideas as to my error? I do free the allocated array as the last step.

As always,
Ed Wright

Subject: Re: Returning a string array from a DLM
Posted by Randall Skelton on Wed, 18 Dec 2002 12:38:30 GMT
View Forum Message <> Reply to Message

Hi Ed,

I hope by now one of the helpful RSI Christmas elves have answered your question privately. In the event that they haven't, the following may be of help. If you don't already own a copy of Ronn's book, I highly recommend it. Ronn basically goes through all the basic types and gives you a few examples for each. With this and the EDG in hand, you can figure out how to make some relatively complicated, C functions.

http://www.kilvarock.com/books/callingCfromIDL.htm

Without seeing your code, it is almost impossible to diagnose what you have done wrong. I suspect you are simply trying to free something that should not be freed... Attached are some intermediate examples I was able to strip out of some of my code. I haven't included any really simple examples as these are all well covered by Ronn's book.

The following examples:

```
IDL> TEST_Parse, 'Hello there; hi; hi; hi; hi; hi; hi; hi; hi; hi', ';', a
IDL> string = TEST_GetStrings(findgen(10))
IDL> string = TEST_CreateStrings(indgen(10,10))
```

The latter two are only subtly different and basically show alternative ways of dealing type conversion and IDL temporary variables. Both, however, show how to use MakeTempArray for IDL_TYP_STRING and use IDL_StrStore. The first example shows how I would parse off a string with a given deliminator. Essentially, you need a ragged 2d array for the strings which, in C, means pointers to pointers unless you use static arrays (poor idea).

Cheers,
Randall

```
-- Start: test.dlm --
MODULE TEST2
DESCRIPTION Testing passing a string array
VERSION 1.0
SOURCE Randall Skelton <skelton@atm.ox.ac.uk>
BUILD_DATE November 2002
FUNCTION TEST_CREATESTRINGS 1 1
FUNCTION TEST_GETSTRINGS 1 1
PROCEDURE TEST_PARSE 3 3
-- End: test.dlm --


-- Start: test.c --
/* ANSI */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* IDL */
#include "export.h"

/* Definitions */
#define TEST_ERROR   0
#define STR_LEN   32
#define TOKEN_ALLOC   8

/* Useful macros */
#define ARRLEN(arr) (sizeof(arr)/sizeof(arr[0]))

/* Protos */
extern IDL_MSG_BLOCK msg_block;
extern void TEST_exit_handler(void);
extern int  TEST_Startup(void);
extern IDL_VPTR IDL_CDECL TEST_GetStrings(int argc, IDL_VPTR argv[], char *argk);
extern IDL_VPTR IDL_CDECL TEST_CreateStrings(int argc, IDL_VPTR argv[], char *argk);
extern void IDL_CDECL TEST_Parse(int argc, IDL_VPTR argv[], char *argk);

/* Messages */
static IDL_MSG_DEF msg_arr[] = {
  { "TEST_Error", "%NError: %s." },
 };

IDL_MSG_BLOCK msg_block;

int IDL_Load(void) {
 /* Define the message block */
 if (!(msg_block = IDL_MessageDefineBlock("TEST", ARRLEN(msg_arr),
```

```
    msg_arr))) {
  return IDL_FALSE;
 }

 /* Call the startup function to add the routines to IDL. */

 /* Routines */
 if (!TEST_Startup()) {
  IDL_MessageFromBlock(msg_block, TEST_ERROR,
   IDL_MSG_RET, "Unable to initialize TEST interface");
 }
 return IDL_TRUE;
}

#ifdef __IDLPRE53__
/* FOR IDL < 5.3 */
/* Define the functions */
static IDL_SYSFUN_DEF TEST_functions[] = {
      {TEST_GetStrings, "TEST_GETSTRINGS", 1, 1, 0},
      {TEST_CreateStrings, "TEST_CREATESTRINGS", 1, 1, 0},
};
static IDL_SYSFUN_DEF TEST_procedures[] = {
      {(IDL_FUN_RET) TEST_Parse, "TEST_PARSE", 3, 3, 0},
};
#else
/* FOR IDL >= 5.3 */
/* Define the functions */
static IDL_SYSFUN_DEF2 TEST_functions[] = {
      {TEST_GetStrings, "TEST_GETSTRINGS", 1, 1, 0, 0},
      {TEST_CreateStrings, "TEST_CREATESTRINGS", 1, 1, 0, 0},
};
static IDL_SYSFUN_DEF2 TEST_procedures[] = {
      {(IDL_FUN_RET) TEST_Parse, "TEST_PARSE", 3, 3, 0, 0},
};
#endif


/* Startup call when DLM is loaded */
int TEST_Startup(void)
{

/* If IDL is pre-5.3 then change IDL_SysRtnAdd to IDL_AddSystemRoutine,
 * (NB: the parameters stay the same) */

#ifdef __IDLPRE53__
 /* FOR IDL < 5.3 */
   /* Add functions */
    if (!IDL_AddSystemRoutine(TEST_functions, TRUE,
```

```
 ARRLEN(TEST_functions))) {
       return IDL_FALSE;
    }
 /* Add procedures */
 if (!IDL_AddSystemRoutine(TEST_procedures, FALSE,
  ARRLEN(TEST_procedures))) {
     return IDL_FALSE;
 }
#else
 /* FOR IDL >= 5.3 */
   /* Add functions */
    if (!IDL_SysRtnAdd(TEST_functions, TRUE,
  ARRLEN(TEST_functions))) {
       return IDL_FALSE;
    }
    /* Add procedures */
 if (!IDL_SysRtnAdd(TEST_procedures, FALSE,
  ARRLEN(TEST_procedures))) {
     return IDL_FALSE;
 }
#endif

    /* Register the error handler */
    IDL_ExitRegister(TEST_exit_handler);
    return(IDL_TRUE);
}

/* Called when IDL is shutdown */
void TEST_exit_handler(void) {
    /* Nothing special to do in this case */
}


 /*-------------------------------------------------------- ------------------*/
/* Procedure: TEST_Parse
 *
 * Purpose:  Parse a string into an array
 *
 * Called in IDL as:
 *    TEST_Parse, string, separator, return
 *
 * Where:
 *    string = A string to parse.
 *    separator = A string separator
 *    return = The returned array
 *
 */
void IDL_CDECL TEST_Parse(int argc, IDL_VPTR argv[], char *argk) {
```

```
/* Local */
IDL_VPTR ret_str;   /* The returning string array */
IDL_STRING *ret_ptr;   /* Pointer to the returning data block */
long n;     /* The number of strings in return array */
long i;     /* A general index */
char **token;    /* Pointer to a Pointer to the parsed string (2D) */
char *parsed;    /* Pointer to the parsed string */
IDL_MEMINT dim[IDL_MAX_ARRAY_DIM]; /* Array dimension */

/* The first passed argument is the string to be parsed.  Ensure
 * it is a scalar string. */
IDL_ENSURE_SCALAR(argv[0]);
IDL_ENSURE_STRING(argv[0]);

/* The second passed argument is the separator. Ensure it is a
 * a scalar string. */
IDL_ENSURE_SCALAR(argv[1]);
IDL_ENSURE_STRING(argv[1]);

/* Ensure we can write to the third passed argument */
IDL_StoreScalarZero(argv[2], IDL_TYP_STRING);

/* At this point our problem is that we don't know how many
 * strings we need to create and therefore do not know how
 * to size the IDL_MakeTempArray call.  We begin by allocating
 * a fixed number of scratch space via a 2D ragged string array.
 * As we proceed, we count the number of rows and determine if we
 * need to allocate more space. */

/* Ensure we initialize the total number of parsed strings to zero */
n = 0;

/* Allocate the first block of space for the token */
token = (char **) malloc(sizeof(char**) * TOKEN_ALLOC);

/* Initialize strtok with the string in argv[0] */
parsed = strtok(argv[0]->value.str.s, argv[1]->value.str.s);

/* While statement */
while ( parsed ) {

  /* Catch the first instance */
  token[n++] = parsed;

  /* Allocate more memory as required */
     if (!(n%TOKEN_ALLOC))
 token = (char **) realloc(token,(n+TOKEN_ALLOC)*sizeof(char **));
```

```
  /* Get the next token */
  parsed = strtok(NULL, argv[1]->value.str.s);
 }

 /* Assign the number of parsed as the number of elements in the
  * first (and only) dimension */
 dim[0] = n;

 /* Create a temporary IDL array to place the output into */
 ret_ptr = (IDL_STRING *) IDL_MakeTempArray(IDL_TYP_STRING, 1, dim,
  IDL_ARR_INI_NOP, &ret_str);

 /* Copy the parsed strings into the ret_str data block */
 for(i=0; i<n; i++) IDL_StrStore(&(ret_ptr[i]), token[i]);

 /* Free the locally allocated memory */
 free(token);

 /* Copy ret to argv[2] */
 IDL_VarCopy(ret_str, argv[2]);

}

 /*-------------------------------------------------------- ------------------*/
/* Function: TEST_GetStrings
 *
 * Purpose:  Returns a string for each passed value. Vectorized.
 *
 * Called in IDL as:
 *    string = TEST_GetStrings(value)
 *
 * Where:
 *    value = an IDL scalar or array of any type other than a structure or
 *            file handle.
 *    string = the returned string(s).
 *
 */
IDL_VPTR IDL_CDECL TEST_GetStrings(int argc, IDL_VPTR argv[], char *argk) {

 /* Local */
   IDL_VPTR in, ret; /* The input and return IDL variables */
 IDL_STRING *in_data; /* A pointer to the input data block */
 IDL_STRING *ret_data; /* A pointer to the returning data block */
 char buffer[80]; /* A static character buffer */
 IDL_LONG n;  /* The number of elements in the passed array */
    long i;   /* General loop index */
```

```
/* The first passed argument is our input */
in = argv[0];

/* If the input is not a STRING then convert it, creating a
 * temporary variable. IDL_BasicTypeConversion() will return the
 * input value without creating a temporary variable. Structures
 * and file handles will cause an IDL_LONG_JMP error. */
in = IDL_BasicTypeConversion(1, argv, IDL_TYP_STRING);

/* Obtaine a pointer to the variable data and count the number
 * of passed elements. */
IDL_VarGetData(in, &n, (char **) &(in_data), TRUE);

if (in->flags & IDL_V_TEMP) {

 ret = in;
 ret_data = in_data;

} else {

  /* Determine if we passed an array */
  if (in->flags & IDL_V_ARR) {

   /* Input is not a temporary variable so allocate new temporary storage */

    /* Allocate a temporary array attached to the ret variable with a
     * data block pointing to ret_data. We use the same dimensions
  * as were passed in the input. */
    ret_data = (IDL_STRING *) IDL_MakeTempArray(IDL_TYP_STRING,
  in->value.arr->n_dim, in->value.arr->dim, IDL_ARR_INI_NOP, &ret);

  } else {

    /* Allocate a temporary variable to ret */
    ret = IDL_Gettmp();
    ret->type = IDL_TYP_STRING;
    ret_data = (IDL_STRING *) &(ret->value.c);

  }
}

/* We now have storage for the string whether it be an array or a scalar. */

/* Create a string on the C side to pass back */
for (i=0; i<n; i++) {

  /* Copy a string into IDL */
  sprintf(buffer, "You passed the value '%s'", in_data[i].s);
```

```
   IDL_StrStore(&(ret_data[i]), buffer);

 }

 return (ret);
}

 /*-------------------------------------------------------- ------------------*/
/* Function: TEST_CreateStrings
 *
 * Purpose:  Creates some strings in C.
 *
 * Called in IDL as:
 *    string = TEST_GetStrings(value)
 *
 * Where:
 *    value = an IDL scalar giving the number of strings to return.
 *    string = the returned string(s).
 *
 */
IDL_VPTR IDL_CDECL TEST_CreateStrings(int argc, IDL_VPTR argv[], char *argk) {

 /* Local */
   IDL_VPTR in, ret; /* The input and return IDL variables */
 IDL_LONG *in_data; /* A pointer to the input data block */
 IDL_STRING *ret_data; /* A pointer to the returning data block */
 IDL_LONG n;  /* The number of elements in the passed array */
 char buffer[80]; /* A static character buffer */
     long i;   /* General loop index */

 /* The first passed argument is our input */
 in = argv[0];

 /* If the input is not a STRING then convert it, creating a
  * temporary variable. IDL_BasicTypeConversion() will return the
  * input value without creating a temporary variable. Structures
  * and file handles will cause an IDL_LONG_JMP error. */
 in = IDL_BasicTypeConversion(1, argv, IDL_TYP_LONG);

 /* Obtaine a pointer to the variable data and count the number
  * of passed elements. */
 IDL_VarGetData(in, &n, (char **) &(in_data), TRUE);

 /* Determine if we passed an array */
 if (in->flags & IDL_V_ARR) {

   /* Allocate a temporary array attached to the ret variable with a
    * data block pointing to ret_data. We use the same dimensions as
```

```
    * were passed in the input. */
   ret_data = (IDL_STRING *) IDL_MakeTempArray(IDL_TYP_STRING,
    in->value.arr->n_dim, in->value.arr->dim, IDL_ARR_INI_NOP, &ret);

 } else {

   /* Allocate a temporary variable to ret */
   ret = IDL_Gettmp();
   ret->type = IDL_TYP_STRING;
   ret_data = (IDL_STRING *) &(ret->value.c);

 }

 /* We now have storage for the string whether it be an array or a scalar. */

 /* Create a string on the C side to pass back */
 for (i=0; i<n; i++) {

   /* Copy a string into IDL */
   sprintf(buffer, "Element %li is %li", i, in_data[i]);
   IDL_StrStore(&(ret_data[i]), buffer);

 }

 /* Unallocate the temporary storage we needed for type conversion */
 IDL_DELTMP(in);

 return (ret);
}

-- End: test.c --
```

On Thu, 12 Dec 2002, Ed Wright wrote:

> Question for DLM experts:
>
> I wrote a DLM for a C routine that returns an array of strings. The first
> version of the DLM code has the return array declared as char item[X][Y].
> After the C call, I created a temp array of strings via a IDL_MakeTempArray
> call, then looped over each string in the array copying the string to the
> array created in IDL_MakeTempArray via an IDL_StrStore call.
>
> It worked..
>
> So being quite industrious, I decided it a good idea to dynamically allocate
> the memory required for the string array. After a few hours of aggravation
> and a phone call to a friend who reminded me a pointer pointing to nothing

> might cause some problems, I have a DLM than can dynamically allocate the
> needed space for a string array then pass that array to my C routine as an
> argument for return.
>
> Now the problem. On return to the IDL interpreter, I experience a plethora
> of malloc errors:
>
> Given a script making two calls to the DLM, call format:
>
> cspice_lparse, string_with_separator, separator_string, item
>
> Item should return as an array of strings.
>
> My test output -
>
> IDL> xtspice_idl
> Start tests
> % Loaded DLM: COMBINE.
> Start of module: st01_idl
> In C routine:
> String: a
> String: b
> String: c
> In IDL, array = a b c
>
> In C routine:
> String: a
> String:
> String: c
> *** malloc[3990]: error for object 0x29cee0: Incorrect check sum for freed
> object - object was probably modified after beeing freed; break at
> szone_error
> Segmentation fault
>
> Any ideas as to my error? I do free the allocated array as the last step.
>
> As always,
> Ed Wright
>
>
>
>
>