Subject: Array has a corrupted descriptor when destroying a Dicom Object
Posted by muswick on Mon, 29 Apr 2002 05:44:10 GMT
View Forum Message <> Reply to Message

I just came accross this problem during the past week, and I have verified this to be a problem on Windows IDL versions 5.3 and 5.5, Windows NT and Windows 2000 Professional and Server OSes.

Any work arounds for the leaking memory would be appreaciated until RSI can come up with a fix.

example code:

```
 ------------------------------------------------------------ -----------------
IDL> f = 'CT.DCM'
IDL> oDcm3 = OBJ_NEW('IDLffDicom',f)
IDL> oDcm3->DumpElements
...
  69 : (0028,0010) : US : IMG Rows : 2 : 512
  70 : (0028,0011) : US : IMG Columns : 2 : 512
...
 140 : (7FE0,0010) : OW : PXL Pixel Data : 525312 : 25 25 25 25 25 25
25 25 ...


IDL> Image = oDcm3->GetValue('7fe0'x, '0010'x)
IDL> help,image
IMAGE           POINTER   = Array[1]
IDL> help,*image[0]
<PtrHeapVar3964>
               UINT      = Array[512, 512]

IDL> help,/memory
heap memory used:   15273998, max:   15274029, gets:    25272, frees:
  22693
IDL> OBJ_DESTROY,oDcm3
% Array has a corrupted descriptor: <No name>.
% Execution halted at: $MAIN$
IDL> OBJ_DESTROY,oDcm3
IDL> HEAP_GC, /VERBOSE
<PtrHeapVar4240>
               STRUCT    = -> IDLFFDICOMELEMENT Array[141]

IDL> help,/memory
heap memory used:   15257497, max:   15273998, gets:    25274, frees:
  22815
 ------------------------------------------------------------ -----------------
```

I can work around the error using catch, but I can't work around the

memory leakage. I went as far as getting a reference to the pointer (using
PTR_NEW(###, /CAST) that contains the pixel data, and using PTR_FREE to get rid of it. The PTR_FREE resulted in the same error.

The problem is that the DICOM file has 1024 bytes additional data than the ROW/COL sizes would indicate: 512x512x2=524288 bytes versus 525312 bytes indicated by the DumpElements. Thus when the IDLffDicom object read the file, I am guessing it most likely created room for 525312 bytes of data, but when it attempted to convert the memory area to an array, the excess data which is not accessible was left behind, along with the original memory size pointer. When IDL attempts to free the memory, the 512x512 UINT doesn't match the memory size pointer, the error is generated, but not freeing memory. They probaly took this approach to avoid the software from crashing.

Ideally, it would be nice if the Dicom file had the correct size in the image area to match the ROWxCOLxBYTES parameters. But from Part 5 sec 8.1.1 of the Dicom standard indicates that:

"This means that the Value Field may need to be padded with data that is not part of the image and shall not be considered significant."

and

"Note: Receiving applications should be aware that some older applications may send Pixel Data with excess padding, which was not explicitly prohibited in earlier versions of the Standard. Applications should be prepared to accept such Pixel Data elements, but may delete the excess padding. In no case should a sending application place private data in the padding data."

Also, with any compressed transfer syntaxes, the length of the pixel data area will not match the image size paramenters. Thus RSI needs to be able handle this situation.

My application continues to function if I use CATCH to handle the error, but reading in excess of 150 plus CT images results in over 75 Mbytes of memory leakage each time a user reads in a new volume of data.

I could use an alternate DICOM Reader, but IDL code based readers are much slower (and with hundreds of images, it gets significant) and there is no way to detect this type of file upfront, to choose a different reader for just these files.

Any thoughts or suggestions would be appreciated.

Gary Muswick
University Hospitals of Cleveland

PS, These images were transferred from a FUJI Synpase Archive.