Subject: HANDLE_FREE: when to use? is it necessary?
Posted by Russ Welti on Mon, 22 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

I was told once by RSI that if one is only about to reassign
a handle to point to a NEW piece of data (when it already
points to some other data) it is *not* necessary to call
HANDLE_FREE.

Also, interestingly, using grep in idl's home /lib dir and in
several pub domain dirs which have hundreds of IDL source files
and applications, I can find only 3 or 4 occurrences of 'handle_free'
OR 'HANDLE_FREE'. Why does it not get used? Why does the manual
never say or show how to use it? Is it really so unimportant?

I ask because my app is getting memory hungry, and its memory
consumption currently monotonically grows over the course
of an extended session of using it, even though when user
'opens' a new 'doc', I set all large arrays to scalar 0.

All compiling has been done before I start taking measurements.

The only possible source would seem to be the fact that I am
not explicitly freeing handles, I am simply pointing them to
new things.

If freeing handles is really the answer, and is a required practise,
then why so few occurrences of its use in IDL programs?

Anyone ever gotten a good FAQ on IDL memory management, beyond the
few pointers given in the User's Guide, and what can be gleaned
from the C programming info in the Advanced Dev. Guide??

Thanks much,

```
                                  /
Russ Welti                            /-\
                                 (c-g)
University of Washington              \-/
Molecular Biotechnology                /
PO Box 357730                       /-\
Seattle, WA  98195                  (a-t)
rwelti@u.washington.edu               \-/
(206) 616 5116 voice   (206) 685 7301 FAX       /
```

Subject: Re: HANDLE_FREE: when to use? is it necessary?
Posted by steinhh on Fri, 26 Jul 1996 07:00:00 GMT

In article <Pine.SOL.3.91.960722114953.671B-100000@mapper.mbt.washington.edu>, Russ Welti <rwelti@mapper.mbt.washington.edu> writes:
|>
|> I was told once by RSI that if one is only about to reassign
|> a handle to point to a NEW piece of data (when it already
|> points to some other data) it is *not* necessary to call
|> HANDLE_FREE.
|>
|> Also, interestingly, using grep in idl's home /lib dir and in
|> several pub domain dirs which have hundreds of IDL source files
|> and applications, I can find only 3 or 4 occurrences of 'handle_free'
|> OR 'HANDLE_FREE'. Why does it not get used? Why does the manual
|> never say or show how to use it? Is it really so unimportant?
|>

I find 8 lines with handle_free (case insensitive search), 4 lines
with handle_create, 23 occurrences of handle_value, 15 handle_info
lines and 1 line with handle_move. Pretty good programming practice
by RSI staff, if you ask me. This is for IDL 3.6.1c.

When I started using handles for my own purposes, I did a (too?)
quick check to se what effects handle_free could have. As it turns
out, the handle *numbers* appeared to be always increasing regardless
of handle_free calls, so I figured that RSI just hadn't gotten around
to actually freeing any handles. Still, in my own programs, I insist
on using handle_free whenever an "object" using a handle is
"decommissioned", simply because it's good programming practice,
and because I expected RSI to get it right in some future version.

When reading your questions, I did a further check like this:

Start a fresh IDL session, then run the following:

```
PRO handle_test1,N

  time = systime(1)

  large_array = fltarr(10000)
  null = 0
  time = systime(1)

  FOR i = 0L,N DO BEGIN
     handle = handle_create()
     ;; Leaves large_array undefined
     handle_value,handle,large_array,/set,/no_copy

     ;; Get back large_array, make *very* sure no optimization skips
```

```
    ;; everything
    handle_value,handle+null,large_array,/no_copy

    handle_free,handle
  END

  PRINT,systime(1)-time
END
```

No matter how many times you run this, the timing is quite impressive,
and it stays constant at about 5 seconds for N=100000 on my machine:

```
** Structure !VERSION, 3 tags, length=48:
   ARCH            STRING    'alpha'
   OS              STRING    'OSF'
   RELEASE         STRING    '3.6.1c'
```

Now, comment out the line with handle_free, and run it again.
Try with N=1000 first, you're going to be in for a surprise.
Each time you run the program, the time goes up and up and up.
If you plot the timings, they make an almost straigh line.
Then, run the original version (with handle_free in place).
This will have slowed down considerably, depending on how
many times you executed the non-handle-freeing version, but
now the times will stabilize.

Conclusion: If you're disposing with a handle, then *USE FREE_HANDLE*,
or else you'll be slowing down every application that uses
handles, even if *they* are sticking to good programming practice.

Now, it should be okay to reuse handles without freeing them, as
far as this is done "locally", i.e., you reuse the handle just
about straight away, like in the same piece of code that could
have opted for the use of free_handle.

I guess the problem here is that IDL doesn't keep a list of "free"
handles, but goes through it's allocated handles sequentially to see if
they're free or not. This should be fixed, RSI!! (It might have
been corrected in IDL 4 already, though).

|> I ask because my app is getting memory hungry, and its memory
|> consumption currently monotonically grows over the course
|> of an extended session of using it, even though when user
|> 'opens' a new 'doc', I set all large arrays to scalar 0.
|>

As someone else mentioned, do you use the /no_copy switch?
It *really* speeds up your execution, avoiding large memory

copying operations, and *could* help with the memory growth
isue. You have to be aware of the fact that it
makes the original "source" variable go undefined, though, and
at times you may loose data in crashes (they are recoverable,
though, just go through handles 1,....,handle_create()-1 to see
if they're valid, use handle_info, and try to get at their values).

Other than that, I really think that memory fragmentation is
more to blame than a pileup of handle memory space. Some
OS versions never really free memory once it's allocated to a
given process.

|> If freeing handles is really the answer, and is a required practise,
|> then why so few occurrences of its use in IDL programs?
|>

It's used in more places than handle_create, so the newness
of the whole concept is the cause.

|> Anyone ever gotten a good FAQ on IDL memory management, beyond the
|> few pointers given in the User's Guide, and what can be gleaned
|> from the C programming info in the Advanced Dev. Guide??
|>

There should be one, but I haven't heard of one.

I just pray that RSI will respond to my cry for some decent handle
notation changes, that will make a three-statement thing like

```
  handle_value,handle,val,/no_copy
  result = val * 5.2
  handle_value,handle,val,/set,/no_copy
```

into the following (they can use whatever "special character" or
notation they like, as long as it fits in a single statement).

```
  result = @handle * 5.2 ;;
```

Why they haven't made this available in the first place? I don't
know, but I guess it's attributable to most of them having grown
up with f77. IDL used to be written in Fortran, but now they've
switched to C, from what I hear.

Stein Vidar Haugan

Subject: Re: HANDLE_FREE: when to use? is it necessary?

Posted by chase on Mon, 29 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

>>>> > "Stein" == Stein Vidar (UiO) <steinhh@cdsa2.nascom.nasa.gov> writes:
In article <4tb4cj$dq7@post.gsfc.nasa.gov> steinhh@cdsa2.nascom.nasa.gov (Stein Vidar (UiO)) writes:

Stein> In article <Pine.SOL.3.91.960722114953.671B-100000@mapper.mbt.washington.edu>, Russ Welti <rwelti@mapper.mbt.washington.edu> writes:
Stein> |>
Stein> |> I was told once by RSI that if one is only about to reassign
Stein> |> a handle to point to a NEW piece of data (when it already
Stein> |> points to some other data) it is *not* necessary to call
Stein> |> HANDLE_FREE.
Stein> |>

This makes sense. In this context one can view handles like other IDL
variables. You do not need to free a variable's storage (e.g. by
using the temporary() command) every time you assign a new value to
the variable. Similarly, the freeing of storage a handle points to
happens automatically. So handle_free is unnecessary when you only
want to change the handle's value.

Stein> When I started using handles for my own purposes, I did a (too?)
Stein> quick check to se what effects handle_free could have. As it turns
Stein> out, the handle *numbers* appeared to be always increasing regardless
Stein> of handle_free calls, so I figured that RSI just hadn't gotten around
Stein> to actually freeing any handles.

The handles do appear to get freed (as revealed by handle_info() and
help,/handle). Just because the IDs returned by handle_create() are
not immediately reused does not mean the handle is not freed.

I use IDL version 4.0.1 on various UNIX systems (HPUX, IRIX, SunOS).
"help,/handle" gives some revealing statistics about handles. It
seems that IDL keeps track of handle references using a small hash
table (containing only 421 slots on our systems, but the table
probably only holds top level handles). Using your test program shows
that the table does not grow when using handle_free. However, the
hash table quickly becomes overwhelmed when handle_free is not used
and you create thousands of unique handles. When there are thousands
of active handles chaining is used off of each table slot making the
table very inefficient when you call handle_create (probably for all
other handle functions too).

Because top-level handles are like a global variable there is really
no reason to have thousands of them available simultaneously - it would
be like a C program with thousands of different variables which would
overwhelm most C compilers.

Stein> |> If freeing handles is really the answer, and is a required practise,
Stein> |> then why so few occurrences of its use in IDL programs?
Stein> |>

Use handle_free whenever you will no longer need the handle, e.g.,
when exiting the scope of the reference to the handle. Here is where
the real problem of using handles can get you. If you fail to free
the handle value storage and lose the handle ID then you end up with a
memory leak problem like those that plague C and C++ programmers. IDL
can not know if you have references to a handle since handle IDs are
represented as ordinary long integers rather then by a unique type.
Thus freeing the memory must be done manually by the user.

Chris

--
===============================
Bldg 24-E188
The Applied Physics Laboratory
The Johns Hopkins University
Laurel, MD 20723-6099
(301)953-6000 x8529
chris.chase@jhuapl.edu

Subject: Re: HANDLE_FREE: when to use? is it necessary?
Posted by steinhh on Thu, 01 Aug 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <77g26a8wtl.fsf@custer.jhuapl.edu>, chase@custer.jhuapl.edu (Chris Chase SRM) writes:
|> I use IDL version 4.0.1 on various UNIX systems (HPUX, IRIX, SunOS).
|> "help,/handle" gives some revealing statistics about handles. It

Thanks! I didn't know about the help,/handle switch. It works on
IDL 3.6.1 as well.

|> Because top-level handles are like a global variable there is really
|> no reason to have thousands of them available simultaneously - it would
|> be like a C program with thousands of different variables which would
|> overwhelm most C compilers.
|>

I disagree -- for a handle to act like a global variable, you'd need
a global variable to store the handle number.

You could have, e.g., a linked list, a linked list of lists, or a

tree, or whatever, using thousands of handles. But I guess RSI
people aren't used to that... The C program equivalent would need
one global pointer.

Stein Vidar

Subject: Re: HANDLE_FREE: when to use? is it necessary?
Posted by chase on Mon, 05 Aug 1996 07:00:00 GMT
View Forum Message <> Reply to Message

>>>> > "Stein" == Stein Vidar (UiO) <steinhh@cdsa2.nascom.nasa.gov> writes:
In article <4tqos5$inl@post.gsfc.nasa.gov> steinhh@cdsa2.nascom.nasa.gov (Stein Vidar (UiO)) writes:

Stein> In article <77g26a8wtl.fsf@custer.jhuapl.edu>, chase@custer.jhuapl.edu (Chris Chase SRM) writes:
Stein> |> I use IDL version 4.0.1 on various UNIX systems (HPUX, IRIX, SunOS).
Stein> |> "help,/handle" gives some revealing statistics about handles. It

Stein> Thanks! I didn't know about the help,/handle switch. It works on
Stein> IDL 3.6.1 as well.

Stein> |> Because top-level handles are like a global variable there is really
Stein> |> no reason to have thousands of them available simultaneously - it would
Stein> |> be like a C program with thousands of different variables which would
Stein> |> overwhelm most C compilers.
Stein> |>

Stein> I disagree -- for a handle to act like a global variable, you'd need
Stein> a global variable to store the handle number.

I was comparing only top level handles to global variables, not child
or sibling handles. Only top level handle IDs need to be saved in a
variable.

Stein> You could have, e.g., a linked list, a linked list of lists, or a
Stein> tree, or whatever, using thousands of handles. But I guess RSI
Stein> people aren't used to that... The C program equivalent would need
Stein> one global pointer.

I was mistaken about the handle table. I assumed that "help,/handle"
gave statistics on a table containing only top level handles (which
are like globals). However, after running a test, I found that child
handles (thousands of which would be legitimate and common) seem to
take the same resources in the handle level table. A handle version
of a single linked list with 10,000 elements degrades performance as

much as having 10,000 top level handles.

I would have thought that child handles would be looked up directly
from pointers in parent and sibling handles. Then only top level
handle IDs would need to be in a fast lookup table. [The only reason
for such a table would be for a user interface to lookup top level
handles via a handle ID tag when a direct reference is unavailable.
As I mentioned previously, handles really need to be a distinct data
type like structures and not treated as long integers. In this way
handles references could be tracked and unreferenced handles
garbage collected. In this case, a variable that is a "handle" data
type serves the same function as a handle ID tag.]

Chris

--
==============================
Bldg 24-E188
The Applied Physics Laboratory
The Johns Hopkins University
Laurel, MD 20723-6099
(301)953-6000 x8529
chris.chase@jhuapl.edu

Subject: Re: HANDLE_FREE: when to use? is it necessary?
Posted by steinhh on Wed, 07 Aug 1996 07:00:00 GMT

View Forum Message <> Reply to Message

In article <77rapla4p6.fsf@grant.jhuapl.edu>, chase@grant.jhuapl.edu (Chris Chase SRM) writes:
|> >>>>> "Stein" == Stein Vidar (UiO) <steinhh@cdsa2.nascom.nasa.gov> writes:
|> In article <4tqos5$inl@post.gsfc.nasa.gov> steinhh@cdsa2.nascom.nasa.gov (Stein Vidar (UiO)) writes:
|>
|> Stein> |> Because top-level handles are like a global variable there is really
|> Stein> |> no reason to have thousands of them available simultaneously - it would
|> Stein> |> be like a C program with thousands of different variables which would
|> Stein> |> overwhelm most C compilers.
|> Stein> |>
|>
|> Stein> I disagree -- for a handle to act like a global variable, you'd need
|> Stein> a global variable to store the handle number.
|>
|> I was comparing only top level handles to global variables, not child
|> or sibling handles. Only top level handle IDs need to be saved in a
|> variable.
|>

This is just a minor detail, but if I were to implement e.g., a
binary tree, with structures like

```
  treenode = {TREENODE_STC, LEFT:-1L, RIGHT:-1L, OTHERDATA:<something>}
```

then I'd still use top-level handles for the left and right links,
because that leaves you with an option *not* to allocate a handle
for either the right or left "leg". With the child/sibling scheme,
if you have only one child of a treenode, how do you decide whether
it's the right or the left leg? You'd need to allocate two leg handles
no matter what, and then traverse through the list (a list of
two elements doesn't take long, granted, but still...) to get to
the last one.

There might be other reasons for using the child/sibling feature
if handles, and I'd like to hear about them if anybody knows.
At the present, I just regard them as unnecessary extras that
take up extra space for housekeeping inside the handles.

But even if the left/right legs in the above structure were
top-level handles, they would in no way be global variables.

|>
|> I would have thought that child handles would be looked up directly
|> from pointers in parent and sibling handles.

At the points where you actually use a child handle, it's just a number,
there's nothing a priori that thells you it's the child or sibling
of something, so there's clearly a need to look it up somewhere.

|> As I mentioned previously, handles really need to be a distinct data
|> type like structures and not treated as long integers. In this way
|> handles references could be tracked and unreferenced handles
|> garbage collected. In this case, a variable that is a "handle" data
|> type serves the same function as a handle ID tag.]

Having handles as a separate data type would certainly make life
easier, yes. IDL would use a "reference counter" that was incremented
each time a handle was copied to another variable, and then decrement
the reference counter for that handle each time a variable containing
that handle goes out of scope or is destroyed. Once the reference
counter goes to zero, free the handle. But how would I rewrite
the definition of the above {TREENODE_STC}? We'd at least need
something called a null-handle.

Stein Vidar