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
