

Re: Alternative COBOL "telco" source program

Source: <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2004-05/0952.html>

From: Robert Wagner (*robert.deletethis_at_wagner.net*)

Date: 05/31/04

Date: Mon, 31 May 2004 19:12:40 GMT

riplin@Azonic.co.nz (Richard) wrote:

>*robert.deletethis@wagner.net (Robert Wagner) wrote*

>

>> *The other is the operating system or Cobol runtime handling of called
>> programs. If the caller does a Cancel, Values will be there on the next Call.*

>> *But if the caller forgets or abends or someone else calls your .dll, they
>> probably won't be.*

>

>*If the caller 'forgets' to cancel and expects initial values then it
> is a program bug.*

If program A forgets and program B expects initial values, where is the bug? I think it's in the called program. Systems are more reliable when programs are responsible for their own storage.

>*If the caller 'abends' then it is unlikely that they will notice if or
> not the next call to a program has initial values or not – it won't
> get there.*

If the called program is a .dll or .so, when the caller causes a general protection exception, there is a good chance the OS will not decrement the 'usage count' of the called program. With a non-zero count, it will stay in memory until the OS is rebooted. That's WHY installation packages tell you to reboot.

Even when the usage count goes to zero, Windows will keep DLLs in memory if this registry entry is missing, which it often is, or set to a value other than 1:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\AlwaysUnloadDll

>*If 'someone else calls your .dll' and gets your data values then that
> would be a serious issue, but what systems would this occur with ?*

Windows and Unix systems.

One would hope that a call to GetModuleHandle() followed by a call to FreeLibrary() would reduce the usage count by 1. It's not that easy.

GetModuleHandle() finds only modules linked to your task. If you don't know what it does, calling it to establish a link could be dangerous.

>> *For these reasons, I put 'individual' variables under a group name (rather
>> then using 77 or 01) and initialize it with 'move low-values to
>> unqualified-variables'.*
>
>*One assumes that you only use binary types for these variables.
>Packed and display variables may be just as poorly initialised as
>occurs with ascii zero characters on packed.*

Packed-decimal, as used in Cobol, is a type defined by mainframe hardware. To an Intel assembly language programmer, Packed means multiple integers stored in a single 64-bit word, enabling the hardware to add or multiply up to eight integers in parallel with a single instruction. For example, PADDUSB for "packed add unsigned with saturation, byte". Its most common use is manipulating screen pixels.

Initializing pic x to low-values requires only a shift in thinking about which value means 'empty'.

I use display numbers for DISPLAY on screens, reports and files. There's no reason to use that format for temporary variables in memory.

>> *FWIW, 'pic 1' seems like sufficient information for the compiler to define
>>one bit. It is on the AS/400. If your compiler makes you also write
>>'binary-1', that seems redundant.*
>
>*PIC 1 may (depending on version) be usage display or bit (or other).
>Display may be the default usage, in which case specifying the
>required usage is not redundant.*

Pic 1 was an extension before 2002, so it means whatever the vendor says. It seems pointless to make it default to display. They already had pic 9. Why invent a new type to do the same thing?

A more useful invention would have been SATURATED, meaning the highest value the variable can store. We could turn an indicator on with 'set my-indicator to saturated' and test with 'if my-indicator is saturated'. Syntax would be the same for pic x, 9 or 1. We could also use it for arithmetic 'add a to b on size error set b to saturated'. That's what PADDUSB does.