

Re: Put Variables giving problems

Source: <http://www.tech-archive.net/Archive/VB/microsoft.public.vb.syntax/2005-02/0015.html>

From: Russ Holsclaw (*russ_at_holsclaw.nyet*)

Date: 01/27/05

Date: Wed, 26 Jan 2005 17:17:30 -0700

"Khai" <middae@hotmail.com> wrote in message
news:q4WdnWRNEdnrY2rcRVn-gA@comcast.com...
>I understand the difference between .txt and binary. I was using that so
>I
> could check the file in notepad, since i'm looking specifically for the
> string data. The real extension I use is .iab. I'm not concerned about
> adding to the data structure, as each file will be used once, and by the
> time I add something new, it will only be a simple "jump in word vba,
> make
> change." kinda thing.
>
> this is just a small thing – which will be run by me. Maintenance is not
> a
> problem, but thank you for pointing it out. In the future, I would make
> it
> much more defined and better for the General Public. =)
>
> So – if VB leaves uninitialized strings as full of nulls, then I should go
> ahead and initialize with MA.prop = "" for each string field, including
> the
> array? I think I do that already (source is at home right now.)
>

If you've really defined everything in the type in terms of fixed-size datatypes, it should make no difference whether the elements are initialized or not.

An all-fixed type may not have:

- * Variable length strings (variables defined with "As String" — fixed strings in the format "As String * n" are fixed. By the way, "As String" is NOT equivalent to "As String * 1". The former is a variable string, the latter is a fixed string of one character.
- * No dynamic arrays (arrays declared with nothing inside the parentheses, such as "codes() as String*2" — dynamic arrays are written to disk with array-descriptors appended, identifying the boundaries of the array.
- * No Variant variables (i.e. no variables not declared "As" anything —

microsoft.public.vb.syntax: Re: Put Variables giving problems

variants are written to binary and random files using a "variant descriptor".

If you have ANY of these things, the Len() function will NOT give you the correct value for your disk record size, regardless of the state of the variables. The documentation is not explicit about this. It was written by someone who assumes you know what you're doing.

- > *Since it's Random, isn't it better to use the Len param of the Open line,*
- > *but instead of using Len(MA), perhaps use Len(newInitializedMyDataType) ?*
- > *I*
- > *was under the impression that the MA would be the same size as an*
- > *initialized UDT, since the Strings are all fixed, and the rest of the*
- > *Singles, Dates, etc have specific bytes assigned anyway? When my 2nd*
- > *program runs that would use this file as a record source, I'm planning on*
- > *having it read the records into an array, then, parsing the info, and*
- > *modifying one of the string fields of my UDT, and putting it back in the*
- > *file that it came from. After all records are parsed, that file really*
- > *isn't needed anymore for anything other than cursory glances that*
- > *potentially could never happen, since it's also written to an Excel*
- > *spreadsheet when it's parsed.*
- >
- >
- > *So – as an apprentice to one who's much more wise in the ways, which way*
- > *do*
- > *you think I should take this? Binary, Random, or learn XML for record*
- > *storage?*

Random will probably work, if you're careful. Binary isn't needed if you only have one record type, in which case all records will be fixed-length after all.

Working with binary files that have mixed contents is a good skill to have, but it's not something you do without giving a lot of thought to how the data is to be written, because you have to be able to write a program that can read it and make sense of what it sees. Also, you may have to do special things if you want to write a program to make updates to the data after it has been initially written. Working with such files helps you to appreciate and understand special software such as database management systems, graphic file formats, and other such useful things.

XML is very fashionable right now, but it's extremely inefficient, both in terms of space and processing performance. It looks good on your resume, especially if you're applying for a job where fashion is more important than intelligence. It is only appropriate for "public interchange" of data, where more than one application generates the data and/or more than one program is processing it, and not always even in those cases. ...or for actual *document* files (text written in natural language) where there is an advantage to being able to "tag" certain passages in the text for special handling. The document use is what it was originally designed for. Using XML for pure structured data is usually an enormous waste.

In my 38 years in this business, I have seen many types of "self-descriptive" file formats come into fashion and go out again, but none of them eliminate the need for programmers to actually know what to do with the data. For some stupid reason, some people seem to think that they will be able to write programs that can be thrown any type of data in XML format (or one of the other "self-descriptive" forms), and the program will always "know" what to do with it. This is utter fantasy! At best, XML gives you the ability to ignore data elements that your program wasn't expecting to see, whether it *should* ignore them or not. Big deal! There are plenty of simpler ways of accomplishing that, and all of them are more efficient.

People who sell computer hardware and/or network bandwidth *love* XML.

In this respect, XML is more like a religion than a technology.

...see now you've gotten me started... ;-)

> *Thank you Russ, for you attention to details (and patience, it must be
> wearing thin with me). It's great getting my hands dirty. =D*