

## Re: Discovering variable types...

**Source:** <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2004-08/0900.html>

---

**From:** J French ([erewhon\\_at\\_nowhere.com](mailto:erewhon_at_nowhere.com))

**Date:** 08/08/04

Date: Sun, 8 Aug 2004 07:31:00 +0000 (UTC)

On Sat, 07 Aug 2004 12:46:50 -0400, not@any.adr wrote:

>On Sat, 7 Aug 2004 06:44:17 +0000 (UTC), [erewhon@nowhere.com](mailto:erewhon@nowhere.com) (J French) wrote:

>

>>No - I am a great fan of OverLoad

>>- it is /OverLapped/ in the ReadFile API that shot me in the foot

>

>I know... It messed with my head at first, too.

Personally I reckon that there is a case for breaking access down into two groups, overlapped and non overlapped - but I suppose MS expect us to use wrappers

<snip>

>If you are using my mini memory manager, take a look at the SDK for the heap >calls. You will find a function "HeapSize" which will tell you the size of >any heap block owned by a pointer.

>

> var

> x : pointer;

> begin

> gemem(x,4096);

> writeln(sizeof(x)); // prints "4" on screen.

> Writeln(heapsize(hheap,x)); // prints "4096" on screen.

> end;

>

>This is because X is actually stored in the executable's memory space but the >memory it points to is on the heap.

>

>Now a function similar to HeapSize that would tell me the memory space used by >any given variable, from it's address, would be invaluable in several >circumstances, not the least of which is sizing read and write operations.

>You wouldn't need case statements, just the size of the memory block occupied

>by a variable, as a number.

Yes, I follow, one could achieve that result by having a convention that the first 4 bytes of all (/packed/) Records store their length

>>The 'hit' is not just in the disk head movement, actually the disk  
>>head seldom really moves  
>>- it is in the convoluted pile of tests that takes place as the OS  
>>decides whether the Handle is valid, whether the file is open for  
>>read, whether it is locked  
>  
>Yep.. but it's not so pricey as you might think. Or rather it's a price worth  
>paying at some level, since the last thing you need is carelessness with  
>irreplaceable data.

I only use write buffering when creating 'new' data, and with caching systems /always/ 'write through' the cache immediately

>Double buffering (i.e. creating your own internal buffers) makes lots of sense  
>in a lot of situations and I use it myself. However, I tend to leave that to  
>the application level, not the "primitives" level I'm working on at the  
>moment. Thus I can write buffering schemes that are tailored to the actual  
>needs of each program rather than trying to cook up some universal scheme that  
>isn't actually right for anything... windows has already done that.

That is true, I was looking at this as a generic access approach for sequentially reading data, if one is randomly reading records, then a cache is much more appropriate.

>>I have a hunch that it is something like 6000 times faster pulling  
>>data from memory rather than getting it from a ReadFile

>Given the difference between disk speed and processor speed that's been in  
>evidence since the XT, I'd agree... and the gap is widening all the time.  
>Memory and processor speeds are increasing at a rate something like  
>exponential in relation to storage speed.

And has been for a long time. I remember one simple App that read a file directly into the screen buffer. It worked fine, except on one old PC (I mean PC) fitted with a HardCard disk drive to make it a pseudo XT

The screen remained blank – somehow the data was being read faster than the screen refresh – I had to slow down reading the data.

>The real deal here is custom buffering. For example: my half-completed midi  
>project is what's prompting me to improve disk access. When finished it will  
>pre-load whole songs with a single ReadFile call, parse them from memory and  
>then dispose of the buffers. Were I to do it byte by byte from disk, parsing  
>a 120k midi file (which is rather large for that file type) could take a  
>couple of minutes, from a memory buffer I can do it in about a second. The  
>advantage of doing this at the app level instead of in the wrappers is that  
>another program may need multiple or different sized buffers in which case I'm  
>not restricted by the limitations of the disk access unit.

In this case I would probably just have one pointer moving across the buffer, and coerce that into the appropriate type for the variable

being read – I would prototype it in direct code, but would probably then put it into wrappers

*>Delphi's BlockRead, while ostensibly a good choice for this kind of activity, >does something rather strange that I've not figured out yet. Their "block IO" >subroutines are bizarre beyond belief. On occasion getting the file from disk >into a memory buffer can be \*extremely\* slow... I've seen delays of up to 3 >seconds before disk access occurs, hence: the new file access routines.*

I just can't stand the Delphi (Pascal) filing stuff  
Personally I reckon that it was developed when people still thought of files as 'blocks' – rather than Streams

*>As usual, if something is worth doing it's worth doing right, so I may as well >spend a couple of days getting the whole thing together and have it for life, >rather than putting a work-around in the app and then struggling with this >again in 3 weeks.*

One more set of utility routines in the library – ditto

*>>>I will overload for the base variable types only. Anything else (records, >>>arrays, memory chunks, etc.) will have to go through the bulk io routine... >> >>I would also make all of them call the same underlying reader >>– there is something nice about just having one 'real' routine > >Already done.*

I suspected that – I suspect you are not a fan of Jackson structuring (replicating blocks of code, if there is any difference in them mind you there would be no conditionals ...)

*>>I think you should look at Streams >>– to me the Native Pascal file access is ... well bizarre >>– I think that Borland recognized that >>as did Microsoft when they moved from FCBs to Handles in about MSDOS 2 > >I experimented with TFileStream etc. but in reality these are just wrappers >for Delphi's blockio routines, leaving me with the original problem plus a >potential for new ones. It made more sense for me to just get down and dirty >and do it my way (with apologies to the chairman of the board).*

Yes, they are just wrappers, not for 'blockio' but for 'stream i/o' – they are pretty good wrappers and because they descend from the abstract TStream class, they are very flexible

In your case they are not appropriate, because you are building your own OO from the bareboards up.

*>>Yes that is exactly what I anticipated – also the Boolean, although >>mostly one would not bother to test the results*

>

*>I must be strange then... I always test IO results...*

*>I simply don't trust hardware to do what it's supposed to.*

When reading data sequentially from an open file, there are only two things that can happen

– disk failure/corruption or read past end of file

There is nothing one can do about disk failure

Read past end of file is a programming problem

In both cases your raise exception will handle it

In my case I would get a MessageBox (crude) and an error flag would be set

Where I expect and can handle errors I always test for them explicitly

When the error is totally unexpected, I make the App scream, and then

I bail out entirely

*>>>Easy... just set all the PAS and DCU files to read only.*

*>>>The compiler will fail and your code integrity is preserved.*

*>>*

*>>I would simply change them back from R/O, tinker and continue*

*>>– it is the 'psychological' division that I miss*

*>Hmmm ... sounds about typical.*

*>*

*>I'm more of the "if it ain't broke don't fix it" philosophy.*

I agree, but as one improves one gets itchy with older stuff

What I used to do was to update a local copy of the Library code, and keep that in the Apps directory. I would then get the Linker to use that code explicitly. Gradually more identical local copies would build up, until I was totally confident, at which point the new version would hit the Library and the local versions would get cleaned up.

One can do the same in Delphi ... but only from within Delphi

*>>>For files Nparam holds the record size and Tparam holds the file path.*

*>>>For directory ops Nparam is a filecount and Tparam holds the search wildcards.*

*>>>etc.*

*>>*

*>>That is pretty much what I would go for*

*>>– but I would have different Record types for the File ops and Dir ops*

*>Naaa... I'm not trying to store complex information... just the file/dir*

*>handle, record size and filename/wildcards. Anything else would be overkill*

*>in my situation.*

Dirs and Files are very different animals

>>( I often get slagged off here for recommending using Strings as  
>>buffers – yet a string is just a chunk of memory that knows about  
>>itself. )

>Yep. The only disadvantage of that is that many times the buffers must be of  
>fixed size and ansi strings aren't.

They most certainly are of fixed size, changing the size of a String  
is the near equivalent of ReAllocMem  
– it is done by the programmer not by the system

>Again, the "do it my way" thing comes to the fore and some time ago I created  
>a "wrapped pointer" setup that would let me allocate a chunk of memory with a  
>descriptor block beginning 8 bytes south of the pointer itself. This allows  
>me to keep buffer size and current position right in the memory block. It's  
>really just an expansion on the old pascal strings but it works nicely. (Of  
>course with the new memory manager, I only need to keep 4 bytes for the buffer  
>position. The heap keeps the size parameter for me, so I guess I really  
>should update that as well.)

Keep both – it is a sanity check

>>>I've even got it opening and closing CD–Rom doors!  
>>  
>>Useful ...  
>  
>But a major pain to get working...  
>  
>// open the drive door  
>function FEjectMedia(DName : ansistring) : boolean;  
> var  
> Dstr : ansistring;  
> Fh : longint;  
> br : longword;  
> begin  
> result := false;  
> if fisdrivereal(dname) then  
> if FGetDriveType(dname) in [d\_removable,d\_cdrom] then  
> begin  
> dstr := '\\.\' + Dname[1] + ':';  
> fh := createfile(pchar(dstr),generic\_read,0,  
> nil,open\_existing,a\_normal,0);  
> result := deviceioctl(fh,\$2D4808,nil,0,nil,0,@br,nil);  
> closehandle(fh);  
> end;  
> if not result then  
> exception(FExc\_door\_open\_cmd);  
> end;  
>  
>// close the drive door  
>function FInjectMedia(DName : ansistring) : boolean;

```
> var
> Dstr : ansistring;
> Fh : longint;
> br : longword;
> begin
> result := false;
> if fisdrivereal(dname) then
> if FGetType(dname) in [d_removable,d_cdrom] then
> begin
> dstr := '\\.\' + Dname[1] + '.';
> fh := createfile(pchar(dstr),generic_read,0,
> nil,open_existing,a_normal,0);
> result := deviceioctl(fh,$2D480C,nil,0,nil,0,@br,nil);
> closehandle(fh);
> end;
> if not result then
> exception(FExc_door_close_cmd);
> end;
```

Interesting, annoyingly that only works for NT, 2K XP  
.. unless I'm missing something

```
>>Have a look at the TFileStream, not with a view to using it, but with
>>a view to nicking ideas from it.
>
>Already done.
```

Setting file size is a PITA in Windows  
– did you notice how Delphi wrapped that one

```
>>Ah, and there is one more 'generic' that needs to go with your FGet
>>stuff – that is: ReadDelineatedString
>
>Got you beat! [smile]
>
>function FgetTxt( FH : Fhandle; RData, Dls : ansistring) : longint;
>
>Where DLS = an end marker that can be anything you supply...
>
> BytesRead := FgetTxt(FH,MyStuff,','); // comma delimited text.
> BytesRead := FGetTxt(FH,MyStuff,#12+#10); // Dos text
> BytesRead := FGetTxt(FH,MyStuff,' '); // words
```

I see, you anticipate reading into a buffer  
MyStuff will be either a PChar or a preformatted String

```
>[grin] Your turn...
```

```
FHandle = Record
  DosHandle : LongInt;
  FilePathName : String;
```

```
FileOpenMode : LongInt;  
Buffer : String;  
BufferStartPos : LongInt;  
BufferReadPos : LongInt  
BufferWritePos : LongInt  
BufferDirtyFlag : Boolean  
EofFlag : Boolean <-----
```

For sequential reading of delineated text one may get a zero length string returned, but that signifies nothing  
Raising an exception when reading sequential data is a bit vulgar

```
While Not FH.EofFlag Do  
  Begin  
    FGetTxt( FH, S ); // Default #13#10  
    ProcessData( S );  
  End;
```

Notice I'm still harping on about the buffers.  
Another thing that has occurred to me, all your reading is going to be pretty tight code

```
FH.Buffer := SomeDataFromAnyWhere;  
FH.DosHandle := -1; // Not from true File  
FH.BufferStartPos := 0; //  
FH.EofFlag := False;  
FH.FileSize := Length( FH.Buffer );
```

```
BytesRead := FgetTxt(FH,MyStuff,');
```

Or more simply:  
FOpenBuffer( FH, SomeDataFromAnyWhere );  
BytesRead := FgetTxt(FH,MyStuff,');

And presto, you have a virtual file reading system.

[Grin]

Also there is pure random access:

```
FGetRec( FH, RecNo, ARecord );
```