

## Re: Discovering variable types...

**Source:** <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2004-08/0917.html>

---

*not\_at\_any.adr*

**Date:** 08/08/04

Date: Sun, 08 Aug 2004 12:11:57 -0400

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

I considered adding an Overlapped flag to my open procedures but bailed on the idea since it could seriously mess with program behaviour, especially when doing random access reads. Set the overlapped flag, open a file, read a record and end up working on a blank record... not fun.

About the only place I would consider using overlapped reads and writes would be in a disk buffering procedure where I was doing some intense processing in other threads and could use wait-for-event signaling to tell me the data was ready. Fortunately that particular situation hasn't come up yet.

>>*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 hitch is that Delphi doesn't store static variables (the ones we declare with Var statements on the heap. They're stored in a block at the top of the EXE file itself so that when you load the program you're actually loading the memory allocations for your variables from disk as well. If you notice the Delphi \$M compiler directive lacks the memory allocation variable and now only does stack size. This is why.

The heapsize function only works for dynamic variables (from getmem or new) it's going to return an error with pointers to anything else. Sizeof only works with static variables and is going to return the size of the pointer (4) for dynamics.

Interestingly SizeOf can delve into this memory area and get back the sizes of static variables by name. Looking at the code linked in by a sizeof call it appears to access some kind of table... but I haven't investigated it very far as yet.

>> *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*

I guess I'm lucky that I've not yet been called on to write stuff that needs schemes this complex. I always use writethroughs and never use overlapped access. I'm sure one of these days it will jump up and bark at me, but so far it hasn't.

> *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*

Actually Delphi's more or less done that for us already...

```
Var
  P : pointer;
  x : longword;
  y : somerecord;
begin
  X := longword(P^);
  y := somerecord(P^);
end;
```

Gotta love that little caret!

> *I just can't stand the Delphi (Pascal) filing stuff*

Hence my current project.

> *Personally I reckon that it was developed when people still thought of  
> files as 'blocks' – rather than Streams*

Oh–oh ... so my code is already outdated? [grin]

> *One more set of utility routines in the library – ditto*

Exactly.

> *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 ...)*

Waste of space.

Personal rule: If the same sequence of commands is used twice in a unit, it becomes a function or procedure. If it's used in more than one unit it gets exposed in the Interface section. If it's used in more than one program it gets put into the "popular" unit in my units collection.

I hate spending 6 hours working out some complex subroutine then having to spend another 6 hours 3 months later doing the same job all over again. (Which means this %&##\$ midi file parser I'm working on is going to get carved into stone tablets!)

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

Not exactly... Unless you consider collections of really useful subroutines to be OO.

Other than experimentally, I've never used classes, forms, or objects in any of my code. I don't think that way, the computer doesn't work that way and I never did get comfortable with it. Conceptually it somehow just doesn't make sense to me.

*>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*

Not necessarily... there are situations where one simply doesn't know where the end of the file is... eg. when pulling in text lines it's easier to just keep going till it errors off at the end of the file and assume you got it all.

*>In both cases your raise exception will handle it*

Which, of course, is why it's there.

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

Hence my mods to the SEH mechanism.

*>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*

That would be the smart way.

I, on the other hand, actually use errors as signals for certain events... such as end of file or file not found.

Re: Discovering variable types...

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

LOL... "one improves" ... gees, that never occurred to me...

*>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.*

Interesting. I do something similar by having versional sub-dirs for my collection. I tend to set aside a couple of weeks now and again when I go through and update everything (like now) and the updates will end up in my main "Units" directory. The older versions end up in dated directories under the main one. When I need to use an older unit version, rather than re-write an entire exe I just add the Uses <unit> in <path> construct into the files.

I've even got a couple of units that go all the way back to TP5.0 in there.

*>>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*

Actually the directory access wrapper I've cooked up treats directories as text files, returning the filenames in ansistrings... So, you can actually open a directory and read it as a file without greatly differentiating the activity.

For manipulating attributes, dates, sizes etc. I've added separate calls that can get the attributes of either open or closed files.

For example:

```
procedure showdirectory(P : ansistring);
  Var
    n : ansistring; // filenames
  begin
    FopenDir(FH, '*.*', p); // open directory
    repeat
      N := FGetDir(fh); // get filename
      if n > " then
        begin
          Write(n, tab); // display filename
          writeln(fgetsize(n)); // display size
          ... // do other stuff
        end;
    until n := "";
    fclosedir(FH);
```

end;

I'll probably have to add the more traditional record–return method as well, but for now this is working out very well since 90% of the time all I really care about is the filenames.

```
>>>( 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
```

True.

```
>>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
```

Well... that would save me a half day's worth of updating.

```
>>>>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 := deviceiocontrol(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 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 := deviceiocontrol(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
```

I puzzled over this a bit myself... there are some functions that require an alternative drive identifier "PhysicalDrive<x>" but it's not clear which functions those are. The writeup on the load and eject media functions made no mention of different behaviour by OS at all...

Actually if you, or someone else, could run the createfile and deviceioctl commands from the subroutines above into a win98 machine and see if they work or not, I'd be grateful.

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

Not yet... will take a look though.

```
> 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*

When readfile hits the end of file, it returns 0 bytes and getlasterror returns 0 so it is possible to detect it gently enough... the return when reading an empty string is 1 not 0 bytes. It actually does have to read the null or the size byte to know it's empty. The only time you will get a return of 0 bytes read, without a corresponding error, is at end of file.

An end of file flag might actually be a good idea... thanks.

> *Notice I'm still harping on about the buffers.*

Yes, I did notice that... [smile]

> *Also there is pure random access:*  
>  
> *FGetRec( FH, RecNo, ARecord );*

Finished that part this morning.

FgetRec(FH,RData,RecNum);

also

lock and unlock ... for multiple points of access.

-----  
Laura