Making utilities for ms-dos essay

Making Utilities for MS-DOS

Michael Sokolov

English 4

Mr. Siedlecki

February 1, 1996

Making Utilities for MS-DOS

These days, when computers play an important role in virtually all aspects of

our life, the issue of concern to many programmers is Microsoft’s hiding of

technical documentation. Microsoft is by far the most important system software

developer. There can be no argument about that. Microsoft’s MS-DOS operating

system has become a de facto standard (IBM’s PC-DOS is actually a licensed

version of MS-DOS). And this should be so, because these systems are very well

written. The people who designed them are perhaps the best software engineers in

the world.

But making a computer platform that is a de facto standard should imply a good

deal of responsibility before the developers who make applications for that

platform. In particular, proper documentation is essential for such a platform.

Not providing enough documentation for a system that everyone uses can have

disastrous results. Think of it, an operating system is useless by itself, its

sole purpose is to provide services to applications. And who would be able to

develop applications for an operating system if the documentation for that

system is confidential and available only to the company that developed it?

Obviously, only the company that has developed that operating system will be

able to develop software for it. And this is a violation of the Antitrust Law.

And now I start having a suspicion that this is happening with Microsoft's

operating systems. It should be no secret to anyone that MS-DOS contains a lot

of undocumented system calls, data structures and other features. Numerous books have been written on this subject (see bibliography). Many of them are vital to

system programming. There is no way to write a piece of system software, such as

a multitasker, a local area network, or another operating system extension,

without knowing this undocumented functionality in MS-DOS. And, sure enough,

Microsoft is using this functionality extensively when developing operating

system extensions. For example, Microsoft Windows, Microsoft Network, and

Microsoft CD-ROM Extensions (MSCDEX) rely heavily on the undocumented internals

of MS-DOS.

The reader can ask, “ Why do they leave functionality undocumented?” To answer

that question, we should look at what this “ functionality” actually is. In MS-

DOS, the undocumented “ functionality” is actually the internal structures that

MS-DOS uses to implement its documented INT 21h API. Any operating system must have some internal structures in which it keeps information about disk drives,

open files, network connections, alien file systems, running tasks, etc. And MS-

DOS (later I’ll call it simply DOS) has internal structures too. These

structures form the core of undocumented “ functionality” in MS-DOS. This

operating system also has some undocumented INT 21h API functions, but they

serve merely to access the internal structures.

These internal structures are extremely version-dependent. Each new major MS-DOS

version up to 4. 00 introduced a significant change to these structures.

Applications using them will always be unportable and suffer compatibility

problems. Every computer science textbook would teach you not to mingle with

operating system internals. That’s exactly why these internal structures are

undocumented.

This bring another question, “ Why does Microsoft rely on these structures in its own applications?" To answer this question, we should take a look at an

important class of software products called utilities. Utilities are programs

that don't serve end users directly, but extend an operating system to help

applications serve end users. To put it another way, utilities are helper

programs. Perhaps the best way to learn when you have to mingle with DOS

internals is to spend some time developing an utility for MS-DOS. A good example

is SteelBox, an utility for on-the-fly data encryption. This development project

have made me think about the use of DOS internals in the first place and it has

inspired me to write this paper.

Utilities like SteelBox, Stacker, DoubleSpace, new versions of SmartDrive, etc.

need to do the following trick: register with DOS as device drivers, get request

packets from it, handle them in a certain way, and sometimes forward them to the

driver for another DOS logical drive. The first three steps are rather straightforward and do not involve any “ illicit” mingling with MS-DOS internals.

The problems begin in the last step. MS-DOS doesn’t provide any documented

“ legal” way to find and to call the driver for a logical drive. However, MS-DOS

does have internal structures, called Disk Parameter Blocks (DPBs) which contain

all information about all logical drives, including the pointers to their

respective drivers. If you think of it, it becomes obvious that MS-DOS must have

some internal structures like DPBs. Otherwise how would it be able to service

the INT 21h API requests? How would it be able to locate the driver for a

logical drive it needs to access?

Many people have found out about DPBs in some way (possibly through disassembly

of DOS code). In the online community there is a very popular place for

information obtained through reverse engineering, called The MS-DOS Interrupt

List, maintained by Ralf Brown. This list is for everyone’s input, and the people who reverse engineer Microsoft’s operating systems often send their

discoveries to Ralf Brown, who includes them into his list. The DPB format and

the INT 21h call used to get pointers to DPBs are also in Interrupt List. As a

result, many programmers, including me, have used this information in their

utilities without much thinking.

However, this is not a good thing to do. DPBs exist since the first release of

MS-DOS as IBM PC-DOS version 1. 00, but the DPB format has changed three times

throughout the history. The first change occured in MS-DOS version 2. 00, when

the hard disk support, the installable device drivers and the UNIX-like nested

directories were introduced. The second change occured in MS-DOS version 3. 00,

when the array of Current Directory Structures (CDSs), a new internal structure,

was introduced to support local area networks and JOIN/SUBST commands. The third

change occured in MS-DOS version 4. 00, when 32-bit sector addressing was introduced and an oversight with storing the number of sectors in a File

Allocation Table (FAT) was fixed. The reader can see that each new major MS-DOS

version up to 4. 00 introduced a change in the DPB format. And this is typical

with all MS-DOS undocumented internal structures.

Although one can probably ignore DOS versions earlier than 3. 10, he still would

have to deal with two different DPB formats. And prior to DOS version 5. 00,

where DPBs were finally documented, no one could be sure that a new DOS version

wouldn't change the DPB format once again. In the first version of SteelBox, my

utility that needs to know about DPBs in order to do its work, I simply compared

the DOS version number obtained via INT 21h/AH= 30h with 4. 00. If the DOS version

was earlier than 4. 00, I assumed that it has the same DPB format as IBM PC-DOS

versions 3. 10-3. 30. If the DOS version was 4. 00 or later, I assumed that it has the same DPB format as IBM PC-DOS version 4. xx. However, there are problems with

such assumptions. First, there are some versions of MS-DOS other than IBM PC-DOS,

and some of them have their internal structures different from those of standard

MS-DOS and PC-DOS. For example, European MS-DOS 4. 00 returns the same version

number as IBM PC-DOS version 4. 00, but its internal structuresmuch more closely

resemble that of PC-DOS version 3. xx. Second, prior to Microsoft's documenting

of DPBs in MS-DOS version 5. 00, there was no guarantee that the DPB format

wouldn't change with a new DOS version.

When I was developing a new version of SteelBox, I started to think about how to

use DPBs properly and in a version-independent manner. I justified the use of

DOS internals in the first place because I know that a lot of Microsoft's own utilities use them extensively. The examples are MS-DOS external commands like

SHARE, JOIN, and SUBST, Microsoft Network, Microsoft Windows, Microsoft CD-ROM

Extensions (MSCDEX), etc. Before we go any further, it should be noted that we

mustn't be dumping unfairly on Microsoft. Originally I thought that DOS

internals are absolutely safe to use and that Microsoft doesn't document them

intentionally in order to get an unfair advantage over its competitors. My

reasoning for this was that Microsoft's own utilities have never stopped working

with a new DOS version.

To find the magic of " correct" use of DOS internals, I started disassembling

Microsoft's utilities. First I looked at three DOS external commands, SHARE,

JOIN, and SUBST. All three programs check for exact DOS version number match.

This means that they can work only with one specific version of MS-DOS. This

makes sense, given that these utilities are bundled with MS-DOS and can be considered to be parts of MS-DOS. One of the utilities, SHARE, unlike other DOS

external commands, accesses the DOS kernel variables by absolute offsets in

DOSGROUP, the DOS kernel data segment, in addition to getting pointers to

certain DOS internal structures and accessing their fields. SHARE not only

checks the MS-DOS version number, but also checks the flag at offset 4 in

DOSGROUP. In DOS Internals, Geoff Chappell says that this flag indicates the

format (or style) of DOSGROUP layout (501). If you look at the MS-DOS source

code (I'll explain how to do it in a few paragraphs), you'll see that programs

like SHARE access the kernel variables in the following way:

The kernel modules defining these variables in DOSGROUP are linked in with

SHARE's own modules. Since the assembler always works the same way, the DOS

kernel variables get the same offsets in the SHARE's copy of DOSGROUP as in the

DOS kernel's copy. When SHARE needs to access a DOS kernel variable, it loads the real DOSGROUP segment into a segment register, tells the assembler that the

segment register points to SHARE’s own copy of DOSGROUP, and accesses the

variable through that segment register. Although the segment register points to

one copy of DOSGROUP and assembler thinks that it points to another one,

everything works correctly because they have the same format. The reader can

drawn the following conclusion from this aside: MS-DOS designers have made the

MS-DOS internal structures accessible to other programs only for DOS’own use

(since linking DOS modules in with a program is acceptable only for the parts of

MS-DOS itself).

Having seen that DOS external commands are not a good example for a program that

wants to be compatible with all DOS versions, I turned to Microsoft Network. One of its utilities, REDIR, is very similar to SHARE in its operation. Like SHARE,

it accesses the DOS kernel variables by absolute offsets. I thought that unlike

SHARE, REDIR is not tied to a specific DOS version. Unfortunatelly, I wasn't

able to disassemble it, because as a high school student, I don't have a copy of

Microsoft Network. However, Geoff Chappell says that it has separate versions

for different versions of DOS, just like SHARE. Therefore, I turned to another

utility again.

My next stop was MSCDEX, the utility for accessing the High Sierra and ISO-9660

file systems used by CD-ROMs. Unlike SHARE and REDIR, MSCDEX is not tied to one

specific DOS version. I'm using MSCDEX version 2. 21 with MS-DOS version 5. 00,

but the same version of MSCDEX can be used with PC-DOS version 3. 30. However, it

accesses the DOS kernel variables by absolute offsets in DOSGROUP, just like

SHARE and REDIR. Of course, my question was " How does it do that in a version-

independent manner?" When I disassembled it, I saw that it takes the flag at

offset 4 in DOSGROUP and uses it to determine the absolute offsets of all the

variables it needs. If this flag equals 0, MSCDEX assumes that all offsets it's

interested in are the same as in DOS versions 3. 10-3. 30. If this flag equals 1,

MSCDEX assumes that all offsets it's interested in are the same as in DOS

versions 4. 00-5. 00. For all other values of this flag MSCDEX refuses to load.

Sharp-eyed readers might notice that this check already makes MSCDEX potentially

incompatible with future DOS versions. The comments in the source code for MS-

DOS version 3. 30 (DOSMULT. INC file) refer to MSCDEX, therefore, it had existed

at the time of MS-DOS version 3. 30. It is very doubtful that anyone, including

the author of MSCDEX, could know what offsets would the kernel variables in DOS version 4. 00 have at that time. If this is true, an MSCDEX version that predates

MS-DOS version 4. 00 won’t run under DOS versions 4. 00 and later.

MSCDEX uses the flag at offset 4 in DOSGROUP to determine not only the absolute

offsets of the kernel variables, but also the “ style” of all other DOS internals

that had changed with DOS version 4. 00. My first thought was that I can use this

flag in my utilities when I need to cope with different “ styles” of DOS

internals. However, my next discovery really surprised me and gave me a real

understanding of what I’m doing when I mingle with DOS internals. MSCDEX version

2. 21 refuses to run under DOS versions 6. 00 and later. So much for the idea that

“ Microsoft’s own utilities have never stopped working with a new DOS version.”

In fact, Geoff Chappell refers to this in DOS Internals (501).

The last utility I looked at was Microsoft SmartDrive version 4. 00, which is

bundled with Microsoft Windows version 3. 10. This utility also uses the DOS internal structures, including the version-dependent ones. However, unlike

MSCDEX, SmartDrive doesn’t have a “ top” DOS version number. It compares the DOS

version number with 4. 00 and assumes that DOS similar to versions 3. 10-3. 30 if

it’s lower than 4. 00 and to versions 4. 00-5. 00 if it’s 4. 00 or higher.

SmartDrive assumes that all future DOS versions will be compatible with MS-DOS

version 5. 00 at the level of the internal structures.

The lack of clear pattern in the usage of the undocumented DOS internal

structures by Microsoft’s own utilities made me think seriously about the

possibility of safe use of the DOS internals in the first place. Originally I

thought that Microsoft has some internal confidential document that explains how

to use the DOS internals safely, and that anyone having that magic document can

use the undocumented DOS internals as safely as normal documented INT 21h API.

However, the evidence I have obtained through reverse engineering of Microsoft’s utilities puts the existence of that magic document under question. In

Undocumented DOS Andrew Schulman notes that it is possible that on some

occasions Microsoft’s programmers have found out about the MS-DOS internals not

from the source code or some other internal confidential documents, but from

general PC folklore, just like third-party software developers. For example, the

MWAVABSI. DLL file from the Microsoft Anti-Virus provides a function called AIO_

GetListofLists(). This function calls INT 21h/AH= 52h to get the pointer to one

extremely important DOS internal structure. In the MS-DOS source code this

structure is called SysInitVars. However, in Ralf Brown’s Interrupt List and in

general PC folklore is called the List of Lists. This is an indication that

Microsoft’s programmers sometimes act just like third-party software developers

(Schulman et al., Undocumented DOS, 44).

On several occasions I have made references to the MS-DOS source code. However, most programmers know that the MS-DOS source code is unavailable to non-

Microsoft employees. Therefore, before we go any further, I need to explain how

could I look at the MS-DOS source code. Microsoft gives it to certain companies,

mostly Original Equipment Manufactures (OEMs). Some people can claim that they

are OEMs and get the Microsoft's documents available only to OEMs (however, this

costs a lot of money). And then some people who don't care too much about laws

start distributing the confidential information they have. This is especially

easy in Russia, where copyright laws are not enforced. So one way or another,

knowledge of some parts of MS-DOS source code spreads among the people. The MS-

DOS OEM Adaptation Kit (OAK) contains commented source code for some MS-DOS

modules and include files and . OBJ files made from some other modules.

Let’s summarize what we’ve seen so far. MS-DOS, like any other operating system,

has internal structures. Every computer science textbook would teach you not to

rely on an operating system’s internals. In MS-DOS, the internal structures are

undocumented. Microsoft’s own utilities do rely on them. By reverse engineering

these utilities, looking at the MS-DOS source code, and thinking the problem

through one can come to the conclusion that there is absolutely no safe way of

using the MS-DOS internal structures. The only proper way of using them is not

using them at all.

Not later than I have come to this conclusion, my SteelBox development project

returned me back to the reality. No matter how bad it is to use of the MS-DOS

internals, utility developers like me have to do it because they have no other choice. Now I'm almost sure that this is precisely why Microsoft uses the MS-DOS

internals itself. Before we go any further, I need to clarify one important

detail.

Once a programmer asked Microsoft to document the INT 2Fh/AH= 11h interface,

generally known as the network redirector interface. Microsoft responded:

The INT 2fh interface to the network is an undocumented interface. Only INT 2fh,

function 1100h (get installed state) of the network services is documented.

Some third parties have reverse engineered and documented the interface (i. e.,

" Undocumented DOS" by Shulman sic, Addison-Wesley), but Microsoft provides

absolutely no support for programming on that API, and we do not guarantee that

the API will exist in future versions of MS-DOS.

This sounds like Microsoft saying, " Here's where you get the info, but you

better not use it." (Schulman et al., Undocumented DOS, 495). Some people might think that Microsoft has internal confidential documents describing the MS-DOS

internals much better than Andrew Schulman’s Undocumented DOS, but there are

indications that the MS-DOS source code is the only “ document” Microsoft has

(I’ll address this issue in a few paragraphs). Perhaps the Microsoft’s

programmers themselves use the same documentation as third parties.

So far we have seen that MS-DOS is not a perfect operating system, and it gives

utility developers no other choice but to use its undocumented version-dependent

internals. The reader might ask, “ what can we do about it?” First of all, some

of the former undocumented DOS functionality was documented in DOS version 5. 00.

The reason for that probably was that some INT 21h functions that were used by

DOS external commands like PRINT don’t actually deal with any DOS internals at all, and Microsoft had simply underestimated the usefulness of these functions

originally. Microsoft has even documented the DPBs. However, Microsoft’s

documentation says that the DPBs are available only in DOS versions 5. 00 and

later, but the reader should remember that the DPB format has changed several

times throughout the history. So in this case Microsoft even restricted

themselves in the ability of making changes in MS-DOS by documenting the DPBs.

However, there are still a lot of undocumented internals in MS-DOS. It should be

noted that documenting them is out of question. This would make it impossible to

make significant changes in MS-DOS, thereby stalling its enhancement. In

Undocumented DOS Andrew Schulman suggests that Microsoft could make an add-in to

MS-DOS that would provide “ clean” documented services that would eliminate the

need for the use of DOS internals. Once Microsoft actually did this, when it introduced the IFSFUNC utility in MS-DOS version 4. 00. This utility converted

the “ dirty” and extremely version-dependent redirector interface into a device-

driver-like interface. However, this utility was removed from MS-DOS versions

5. 00 and later (I’ll explain why in a few paragraphs).

Fortunately, the ill-fated IFSFUNC utility was not the only effort to enhance

MS-DOS. In Microsoft Windows versions 3. 00 through 3. 11, there is a component

called Win386. It has got its name from Windows/386, its ancestor. In early beta

releases of Microsoft’s Chicago operating system this component was called

DOS386. When Chicago was renamed into Windows 95, this component was given

uninteresting name VMM32. Finally, the beta release of Microsoft C/C++ Compiler

version 7. 00 included this component from Microsoft Windows under the name

MSDPMI. I think that the best name for this component is DOS386, so I’ll call it this way.

Probably the reader would ask, “ What this component is?” DOS386 is a

multitasking protected-mode operating system. A close inspection of
DOS386

reveals that it has almost nothing to do with Windows, and has a lot to do
with

DOS (that’s why I prefer the name DOS386 over Win386). Two of DOS386’s

subcomponents, DOSMGR and IFSMGR, are perhaps the heaviest users of
DOS

internals. These modules know a lot about the internals of MS-DOS, and they

provide their own interfaces which in fact can help an utility avoid using DOS

internals. For example, let’s return to our SteelBox utility.

This utility needs to access a file from inside an INT 21h call. Most DOS

programmers know that DOS INT 21h API is non-reentrant. It means that no
INT 21h

calls can be made while an INT 21h call is already being serviced. Therefore,
an

utility like SteelBox would have to play tricks with DOS internals with all the consequences. On the other hand, DOS386's IFSMGR subcomponent provides an

interface that replaces INT 21h. Unfortunately, IFSMGR is documented only in the

Windows 95 Device Development Kit (DDK), and I don't have a copy of it yet.

However, it is quite possible that the IFSMGR's interface is reentrant. If it is,

all problems with SteelBox would be immediately solved, and it won't contain a

single undocumented DOS call. Keep in mind, however, that DOS386 is relatively

new, and perhaps its current version doesn't provide all the desired

functionality. But certainly DOS386 is definitely a good foundation for a new

operating system.

Although I definitely don't want to overblame Microsoft, I have to say some

unpleasant truth about this company. In their run for profit, people at

Microsoft violates some principles of free enterprize. In other words, they try

to make a monopoly. One of the unfair things Microsoft does is called

discriminatory documentation. Although the source code for MS-DOS, Microsoft

Network, and other Microsoft products is supposedly unavailable to anyone,

Microsoft has made the source code of some utilities available to selected

vendors (Schulman et al., Undocumented DOS, 495).

Another example is the deliberate incompatibility of some Microsoft products

with Digital Research's DR-DOS. Some programs, including Microsoft
Windows

version 3. 10 beta and Microsoft C Compiler version 6. 00, contain special
code

with sole purpose of making them incompatible with DR-DOS and other DOS

workalikes. Although I'm definitely not a supporter of DOS workalikes, I think

that Microsoft should use fair methods of competition.

Finally, there is a big problem with Microsoft's packaging of MS-DOS and
DOS386.

The most important problem with DOS386 is that it's currently available to
users

only as Win386 in Microsoft Windows. Furthermore, the usual Windows
technical

documentation (except the DDK) doesn't even mention the existence of
Win386,

because it’s actually not a part of Windows. As a result, an amasing number of

programmers don’t even know about DOS386 (or Win386), and many of those how do

greatly underestimate its tremendous importance.

Now Windows 95 comes into play. In this package, MS-DOS, DOS386, and Windows are

thrown into one melting-pot. First of all, the integration of MS-DOS and DOS386

is a very good step. Given the volatility of DOS internals, the DOSMGR

subcomponent of DOS386 (which, remember, is perhaps the heaviest user of DOS

internals) cetainly should be tied to one specific DOS version. However, the tie

between DOS/DOS386 and Windows is largely artificial. Try a simple experiment.

Rename KRNL386. EXE file in your WINDOWSSYSTEM directory into something else,

and put something else (COMMAND. COM fits nicely) into that directory under the name KRNL386. EXE. And then try to run Windows. But instead of running Windows,

this would load and activate Win386 without loading Windows. And there is no

magic in this simple experiment. KRNL386. EXE is the first module of Windows, and

Win386 runs it when it completes its initialization. By putting something else

in place of KRNL386. EXE, one can break the artificial tie between Windows and

DOS386.

At some point of time Microsoft probably throught of making a version of DOS386

which would not be tied to Windows. There was an utility called MSDPMI in the

beta release of Microsoft C/C++ Compiler version 7. 00, which was that very

DOS386 without Windows. But now Microsoft is abandoning MS-DOS and everything

else that is not Windows. Microsoft tries to persuade users that Windows 95

doesn’t contain a DOS component, but this is not true. It is simply tied into

Windows.

Now let’s summarize the above. Microsoft is ignoring the minority users who

don’t like Windows and who want to use MS-DOS and DOS386 without Windows,

because Microsoft cares only about its profit. One person cannot stop them doing

that. Therefore, we, the programmers, should unite. If I call Microsoft alone,

no one would listen to me. But if thousands of us do it together, we might

achieve something. If you have any questions or suggestions about creating an

association of programmers against Microsoft, please send E-mail to Michael

Sokolov at emailprotected

Bibliography

Brown, Ralf. The MS-DOS Interupt List. Not published on paper, available online

for free.

Chappell, Geoff. DOS Internals. New York: Addison-Wesley Publishing Company,

1994.

Microsoft Corporation. Microsoft Windows Device Development Kit. Computer software. Redmond: Microsoft, 1990.

Pietrek, Matt. Windows Internals: The Implementation of the Windows Operating

Environment. New York: Addison-Wesley Publishing Company, 1993.

Schulman, Andrew. , Ralf Brown, David Maxey, Raymond J. Michels, Jim Kyle.

Undocumented DOS: A Programmer’s Guide to Reserved MS-DOS Functions and Data

Structures. New York: Addison-Wesley Publishing Company, 1994.

Schulman, Andrew. , David Maxey, Matt Pietrek. Undocumented Windows: A

Programmer’s Guide to Reserved Microsoft Windows API Functions. New York:

Addison-Wesley Publishing Company, 1992.