

The Windows Concept Journey

Version 3.0
March-2024

By Dr. Shlomi Boutnaru



Table of Contents

Table of Contents.....	2
Introduction.....	5
Win32 API - Working with Strings.....	6
Window Messages.....	7
Recovery Directory.....	8
COM (Component Object Model).....	9
Windows Services.....	11
What IPC (Inter Process Communication) mechanisms do we have in Windows?.....	13
Tasks (Windows Scheduler).....	14
Objects.....	15
Object Manager.....	16
Clipboard.....	17
Recycle Bin.....	18
Handles.....	19
Unnamed Handles.....	20
Processes.....	21
Threads.....	22
Fiber.....	23
Mailslot.....	24
Windows Experience Index (WEI).....	25
File Explorer (previously Windows Explorer).....	26
NTFS (New Technology File System).....	27
Atom Table.....	28

Introduction

When starting to learn Windows I believe that they are basic concepts that everyone needs to know about. Because of that I have decided to write a series of short writeups aimed at providing a basic explanation for fundamental concepts which are part of the Windows operating system.

Overall, I wanted to create something that will improve the overall knowledge of Windows in writeups that can be read in 1-3 mins. I hope you are going to enjoy the ride.

Lastly, you can follow me on twitter - @boutnaru (<https://twitter.com/boutnaru>). Also, you can read my other writeups on medium - <https://medium.com/@boutnaru>. Lastly, You can find my free eBooks at <https://TheLearningJourneyEbooks.com>.

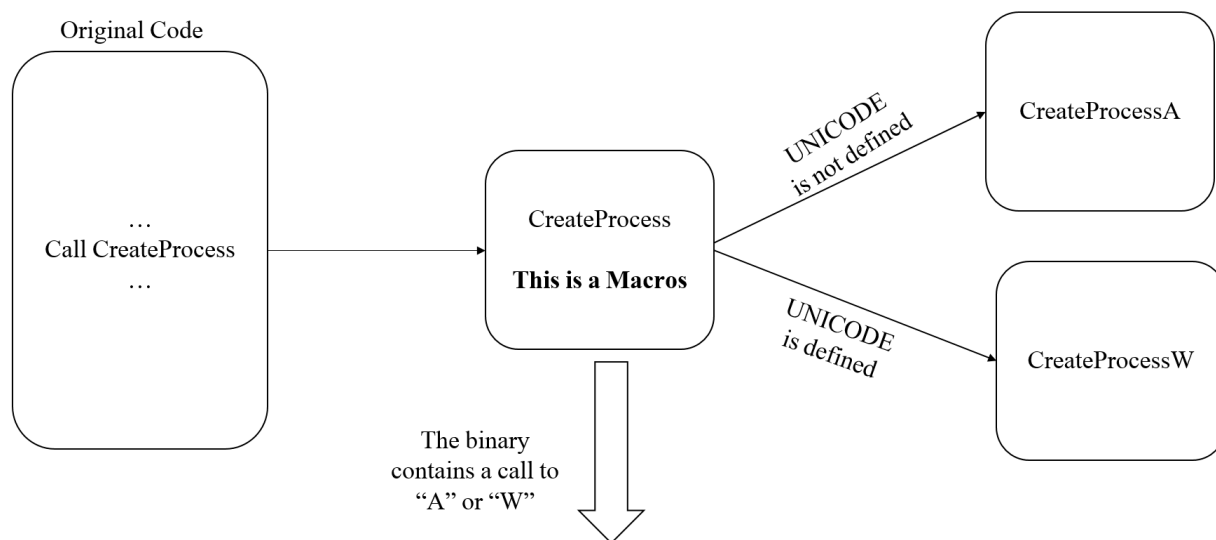
Lets GO!!!!!!

Win32 API - Working with Strings

After Microsoft added support for Unicode as part of Windows, it still needed to support the usage of ANSI strings. The way Microsoft decided to do it is by providing two sets of API (one for ANSI and the second for Unicode). It is important to know that the ANSI version of the API converts the strings to Unicode before calling the relevant syscalls (the kernel is Unicode only).

For example, in order to create a new process we can use "CreateProcessA" or "CreateProcessW" (A for ANSI strings and W for wide char aka Unicode). You might remember that you called CreateProcess and none of the above - so how did it work? The trick is using macros. You called a macro that checked if the UNICODE was defined. If it was defined it made a call to the function ending with "W" else it called the one ending with "A" - see the illustration below for the entire flow.

By the way, in the documentation (like MSDN) the functions names are without the suffix ("A" or "W") despite the fact it is the name of the macro and not of the functions themselves. After compilation the executable contains a reference/dependency to a specific function (you can see it in the diagram below - I have extracted the strings from different DLLs showing the symbol names).



```
C:\Windows\System32>C:\tools\strings64.exe *.dll | findstr /i CreateProcess
C:\Windows\System32\aadtb.dll: DesktopAPI::CallCreateProcess
C:\Windows\System32\aadtb.dll: CreateProcessW
C:\Windows\System32\accessibilitycp1.dll: CreateProcessW
```

Window Messages

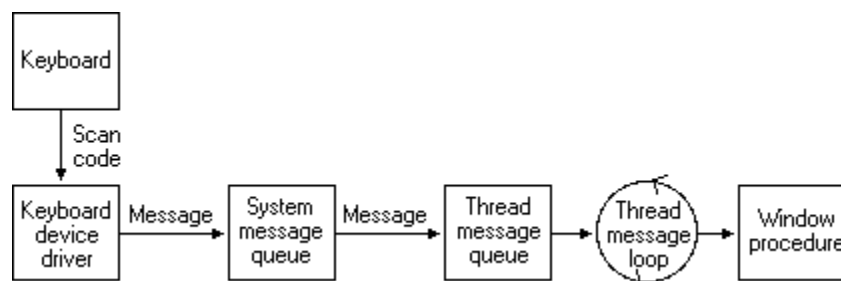
GUI applications under Windows react to different events from the user and the operating system itself. In case of user events, think about: keys pressed, touch-screen gestures, mouse clicks and more. In the case of OS events, think about plugging a new hardware device or changing a power-state¹ (like hibernation or sleep).

Overall, Windows communicates with windows created by applications using messages (aka “Window Messages”). In essence, a message is a number that defines a specific event sent to a window. Every window has a “Window Procedure”, which is a function that processes all messages sent to windows of the same class².

By the way, from the perspective of the Windows kernel the following classes can be created using the “CreateWindow” API call: BUTTON, COMBOBOX, EDIT, LISTBOX, MDICLIENT, SCROLLBAR and STATIC³.

Moreover, due to the fact an application can receive many messages and it can have several windows (each with its own window procedure) a loop to retrieve the message is needed. It is called “The Message Loop” which dispatches the message to the correct window⁴.

To summarize, in a Windows GUI application, “The Message Loop” is a continuous loop that retrieves messages from the operating system and dispatches them to the appropriate “Window Procedure”. It is responsible for handling all of the user input (and other events) that occur in the application - as shown in the diagram below⁵. In order to pull a message from the queue we can call “GetMessage”⁶.



¹ <https://learn.microsoft.com/en-us/windows/win32/learnwin32/window-messages>

² <https://learn.microsoft.com/en-us/windows/win32/winmsg/window-procedures>

³ <http://winapi.freetechnet.com/win32/WIN32CreateWindow.htm>

⁴ <https://learn.microsoft.com/en-us/windows/win32/learnwin32/window-messages>

⁵ <https://learn.microsoft.com/en-us/windows/win32/inputdev/about-keyboard-input>

⁶ <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getmessage>

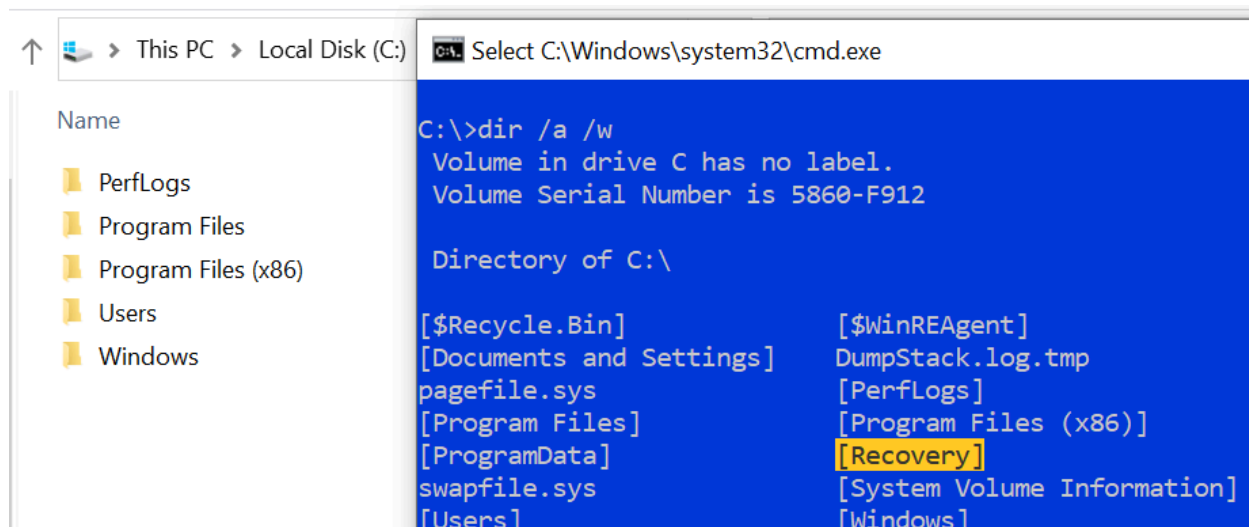
Recovery Directory

A bunch of folks have asked me about what is the goal of different directories in a Windows filesystem hierarchy. So I have decided to write a short series about that. In this writeup we are going to talk about the “Recovery” directory.

It could be that you have never seen this directory before on your root drive (“C:\Recovery”). The reason for that is that the directory is marked “hidden” - as shown in the screenshot below.

By the way it is not enough to display hidden items in explorer to see it. In order to show it we need to unmark “Hide Protected Operating system files (recommended)”⁷ - you can see the entire flow in the following link.

Overall, the directory is a leftover from a previous version of Windows (the version before an upgrade that was made). It is used in cases where there are issues after an upgrade and the user wants to revert back. Thus, after a successful upgrade you can probably delete it⁸.



```
C:\>dir /a /w
Volume in drive C has no label.
Volume Serial Number is 5860-F912

Directory of C:\

[$Recycle.Bin]           [$WinREAgent]
[Documents and Settings]  DumpStack.log.tmp
pagefile.sys           [PerfLogs]
[Program Files]         [Program Files (x86)]
[ProgramData]          [Recovery]
swapfile.sys           [System Volume Information]
[Users]                [Windows]
```

⁷ <https://www.techbout.com/hidden-system-files-windows-10-51145/>

⁸ <https://answers.microsoft.com/en-us/windows/forum/all/can-you-remove-the-large-crecovery-folder-in/e2185d3b-930c-41c0-a1d5-62c753b8a085>

COM (Component Object Model)

COM (Component Object Model) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact with each other. COM is the foundation technology for Microsoft's OLE (compound documents) and ActiveX (Internet-enabled components) technologies. These objects can be within a single process, in other processes, even on remote computers⁹.

COM was introduced by Microsoft in 1993. It is used for IPC (Inter Process Communication) in a variety of programming languages. Also, COM allows the reuse of objects without any knowledge of their internal implementation, it forces component implementers to provide well-defined interfaces that are separated from the implementation¹⁰.

Let us go over a small example of using COM. Excel uses COM to enable users to create/modify/save/share excel files. By using COM we don't need to understand the binary format of excel files in order to perform the different operations. You can see a demonstration for that in the screenshot below.

Moreover, COM objects are registered with the operating system so they could be loaded in the future. The magic behind that is CLSID (Class ID). A CLSID is a globally unique identifier that identifies a COM class object. If your server or container allows linking to its embedded objects, you need to register a CLSID for each supported class of objects¹¹.

CLSID is stored in the registry under HKEY_CLASSES_ROOT\CLSID\{CLSID value}¹². It is used by the operating system to locate the appropriate code for loading. For examples of CLSIDs I suggest going over the following link

<https://www.elevenforum.com/t/list-of-windows-11-clsid-key-guid-shortcuts.1075/>.

They are several related technologies that we are going to talk about in future writeups: COM+, DCOM, Windows Runtime (aka WinRT), XPCOM (aka nano-COM), .NET framework, DEC/RPC, OLE, ActiveX, MSRPC and DDE¹³

⁹ <https://learn.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal>

¹⁰ https://en.wikipedia.org/wiki/Component_Object_Model

¹¹ <https://learn.microsoft.com/en-us/windows/win32/com/clsid-key-hklm>

¹² <https://www.trendmicro.com/vinfo/us/security/definition/clsid>

¹³ <https://learn.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal>

Windows PowerShell

PS C:\tmp> dir

Directory: C:\tmp

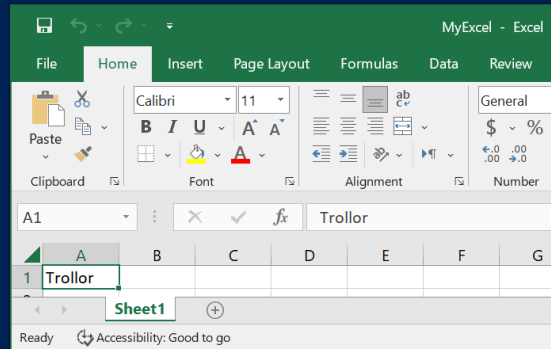
Mode	LastWriteTime	Length	Name
-a----	1/6/2023 10:47 AM	0	empty.txt

```
PS C:\tmp> $MyExcelFile = New-Object -ComObject Excel.Application
PS C:\tmp> $MyWorkbook = $MyExcelFile.Workbooks.Add()
PS C:\tmp> $MySheet = $MyWorkbook.Sheets.Item(1)
PS C:\tmp> $MySheet.Range("A1").Value = "Trollor"
PS C:\tmp> $MyWorkbook.SaveAs("C:\tmp\MyExcel.xlsx")
PS C:\tmp> $MyWorkbook.Close()
PS C:\tmp> $MyExcelFile.Quit()
PS C:\tmp> dir
```

Directory: C:\tmp

Mode	LastwriteTime	Length	Name
-a----	1/6/2023 10:47 AM	0	empty.txt
-a----	1/6/2023 10:48 AM	8668	MyExcel.xlsx

```
PS C:\tmp> start .\MyExcel.xlsx
PS C:\tmp>
```



Windows Services

Before we are going to talk about processes which are related to services handled under Windows (like services.exe and svchost.exe) we have to explain what a service is.

Services are processes which are managed by the operating system, it resembles demons in Linux (but there are a couple of differences that I am going to talk about in a different writeup).

Due to security reasons services can be executed at least under 3 different entities: System, Network Service and Local Service (each of them with different permissions and privileges - we will cover them in more details in the future). Of course, we can also run a service using a local user or a domain user with any access rights that we want.

There are different ways in which we can administer services, however I am going to focus on probably the 4 well known interfaces. First, Win32 API (it is used also for 64 bit despite its name) such as StartService¹⁴. Second, the mmc snap-in “services.msc”¹⁵. Third, PowerShell by leveraging cmdlets such as: New-Service, Get-Service, Restart-Service, Stat-Service, Stop-Service and more. Fourth, the command line tool “sc.exe”¹⁶.

A service can be in one of the three major states: started, stopped or paused. Alos, each service has a startup state which defines what should happen with the service when the OS starts - it could be one of the following: Automatic, Automatic (Delayed Start), Manual or Disabled. Let us go over each and one of them.

Automatic, in this configuration the SCM starts the service as part of the system boot process. In the case of the delayed start, it's an optimization feature to reduce the time it takes the system to boot-up. “Automatic Start” is still run by the SCM but not during the boot process (they are started automatically shortly after the boot process has finished). Manual, in this configuration the SCM does not start the service and it needs to be from some other administrative interface (as we explained above), we can also script it if we want. Disabled, in this configuration even an administrator can't start the service. In order to start the service we first need to enable it by setting it to any setting which is not disabled.

Over the years different security enhancements were added for service hardening (Examples are session isolation, least privileges, restricted network access and service isolation). We are going to speak about it more when talking about security and the process of hardening the OS.

¹⁴ <https://docs.microsoft.com/en-us/windows/win32/services/service-functions>

¹⁵ <https://www.thewindowsclub.com/open-windows-services>

¹⁶ <https://ss64.com/nt/sc.html>

In the screenshot below you can see examples for the things we have talked about regarding the DHCP Client service. On the left we can see the status and the startup type and on the right the user which the service is logging on behalf of.

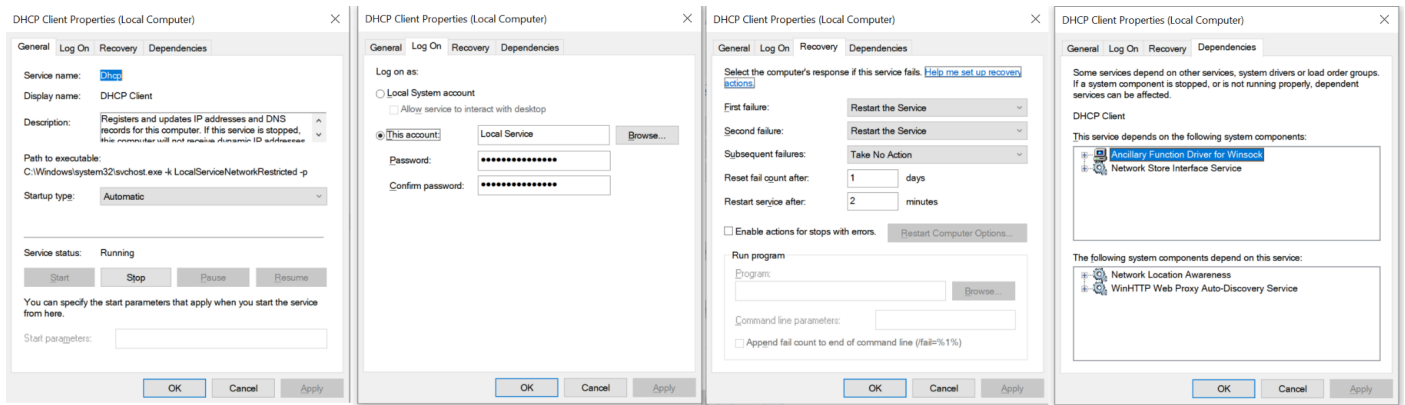
In addition, I am going to talk about: dependency management and recovery handling. In the screenshot below an example of those configurations are shown regarding the “DHCP Client” service (from services.msc). It is time to deep dive into it, so let’s go.

In the case of dependency management in the configuration of each service there is a list of dependent services and the list of the services that depend on it. Those lists are checked by the SCM when starting and stopping services.

Regarding recovery, the configuration allows setting actions for first/second/subsequent failures. The action could be to restart the service/the computer or execute an arbitrary command. We could also reset the fail count after N (we can set it) days and enable actions for cases the service stops with errors.

All the configurations of services are saved as part of the Registry (HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services). It is important to know that kernel drivers configuration is also stored there (we will go over it as part of the SCM discussion).

Next we are going to check out the SCM (Service Control Manager) which is the OS part responsible for managing all the services and their configuration.



What IPC (Inter Process Communication) mechanisms do we have in Windows?

Due to the fact that each process in Windows has its memory address space¹⁷ we can't pass pointers between threads in different processes and expect to see the same data in the same virtual address. It could be that the virtual address is not valid in one of the address spaces or we have a different data stored there (they are of other cases also).

In order to allow different threads (in different processes) to pass data between them we need to use an IPC (Inter Process Communication) mechanism - an illustration of that is seen in the diagram below¹⁸. Moreover, each OS has its own IPC mechanisms. On Windows we have the following mechanisms: clipboard, Windows Messages, COM (Component Object Model), DDE (Dynamic Data Exchange), Shared Memory, File Mapping, Mailslots, Pipes, RPC (Remote Procedure Call), ALPC (Advance Local Procedure Call) and sockets¹⁹.

There are also folks that say we can use files on the filesystem for IPC but it is not a specific mechanism of Windows so I won't speak about it for now. Lastly, when talking about IPC we should also talk about synchronization objects, however I will leave it for a future discussion.

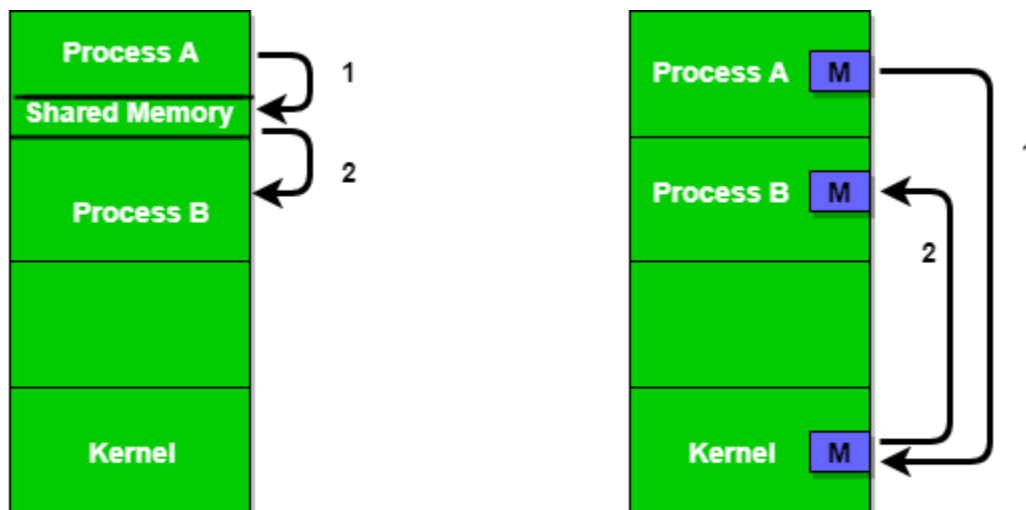


Figure 1 - Shared Memory and Message Passing

¹⁷ <https://medium.com/@boutnaru/linux-memory-management-part-1-introduction-896f376d3713>

¹⁸ <https://www.geeksforgeeks.org/inter-process-communication-ipc/>

¹⁹ https://www.slideshare.net/mfsi_vinothr/ipc-mechanisms-in-windows

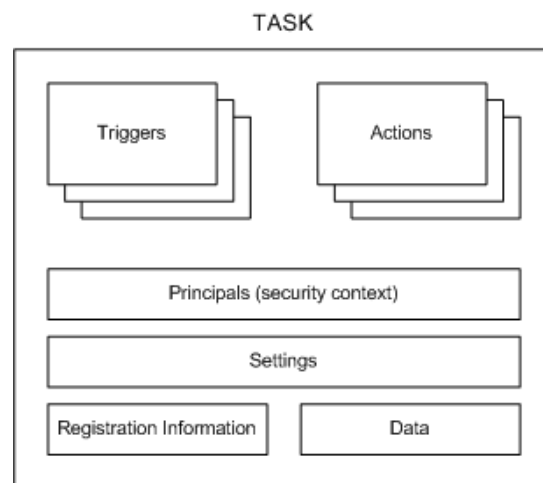
Tasks (Windows Scheduler)

Overall, a task is a scheduled work that is performed by the “Task Scheduler” service. Each task has several components: triggers, actions, principals, settings, registration information and data - as shown in the diagram below²⁰.

Triggers are events/time-based conditions which are used as a criteria for starting an execution of a task. A task can have multiple triggers up to a maximum of 48²¹. Also, actions are the actual work performed by a task. A task can have a single/multiple actions up to a maximum of 32 actions. We can have different types of actions: “ComHandler” (COM), “Exec Action”, “Email Action” (sending an email notification) and “Show Message Action”²².

Moreover, principals is the definition of the security context in which the task is executing on behalf of, including UAC settings and more²³. Settings, that is the configuration used by the “Task Scheduler” while running the task. Think about if we can run multiple instances of the task, or what to do with the task if the system is in idle state and more. By default a task will stop after 72 hours, unless we change the “ExecutionTimeLimit”²⁴.

In addition, registration information is the data collected when the task is created/registered. Data elements that can be included (but not limited to) are: author, date, description, task version, security descriptor and more²⁵. We can also have additional documentation for the tasks (this is the data portion in the diagram shown below). Lastly, “Task Scheduler” has two versions (“1.0” and “2.0”) which have differences in the API they support and the configuration that can be made.



²⁰ <https://learn.microsoft.com/en-us/windows/win32/taskschd/tasks>

²¹ <https://learn.microsoft.com/en-us/windows/win32/taskschd/task-triggers>

²² <https://learn.microsoft.com/en-us/windows/win32/taskschd/task-actions>

²³ <https://learn.microsoft.com/en-us/windows/win32/taskschd/security-contexts-for-running-tasks>

²⁴ https://learn.microsoft.com/en-us/windows/win32/api/taskschd/nf-taskschd-itasksettings-get_executiontimelimit

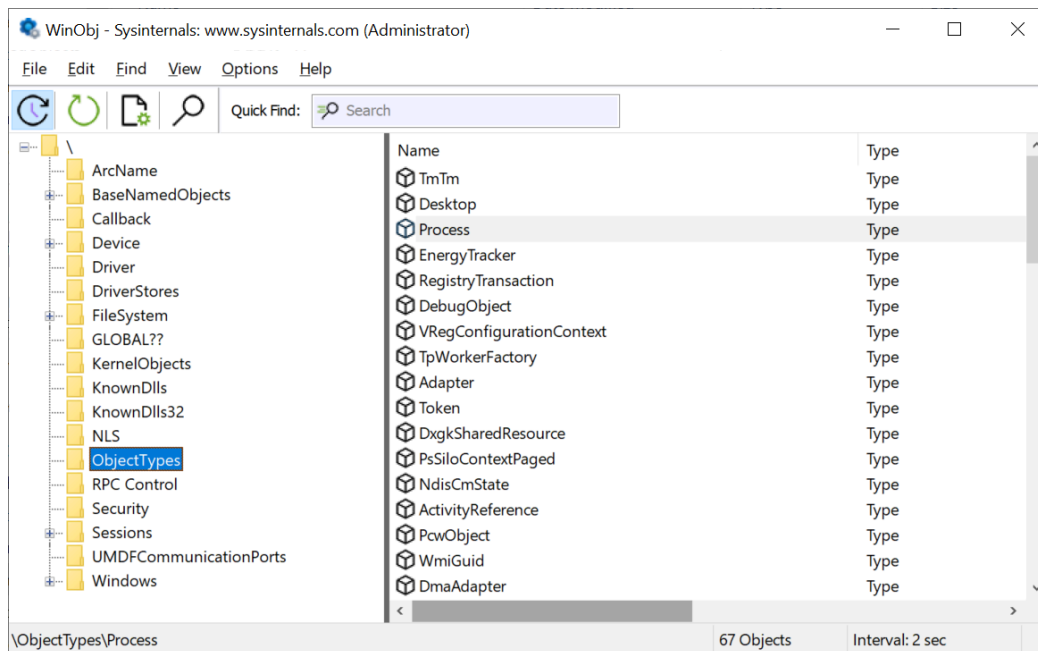
²⁵ <https://learn.microsoft.com/en-us/windows/win32/taskschd/task-registration-information>

Objects

Basically, objects are data structures that represent system resources. Think about things like processes and threads²⁶. Those object are divided to two main parts: the object header “struct _OBJECT_HEADER”²⁷ and the body which holds the specific information regarding a system resource. Examples are: _EPROCESS²⁸, _FILE_OBJECT²⁹ and more.

Moreover, we can see a list of available object types using the “WinObj” tool from Sysinternals - as shown in the screenshot below. By the way, the subsystem that manages the Windows resources is the “Object Manager”, which is part of the “Windows Executive”. The “Windows Executive” is contained as part of “ntoskrnl.exe”³⁰.

Overall, they are three different categories of objects in Windows: user, graphics (GDI objects) and kernel³¹. In User we have objects like: “Hook”³² and “Menu”³³. In GDI we have objects like: “Font”³⁴ and “Region”³⁵. Lastly, in the case of kernel objects we have examples like: “Desktop”³⁶ and “Job”³⁷.



²⁶ <https://learn.microsoft.com/en-us/windows/win32/sysinfo/handles-and-objects>

²⁷ https://www.nirsoft.net/kernel_struct/vista/OBJECT_HEADER.html

²⁸ <https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/ntos/ps/eprocess/index.htm>

²⁹ https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_file_object

³⁰ [https://learn.microsoft.com/en-us/previous-versions/cc768129\(v=technet.10\)](https://learn.microsoft.com/en-us/previous-versions/cc768129(v=technet.10))

³¹ <https://learn.microsoft.com/en-us/windows/win32/sysinfo/object-categories>

³² <https://learn.microsoft.com/en-us/windows/win32/winmsg/hooks>

³³ <https://learn.microsoft.com/en-us/windows/win32/menu/rc/menu>

³⁴ <https://learn.microsoft.com/en-us/windows/win32/gdi/fonts-and-text>

³⁵ <https://learn.microsoft.com/en-us/windows/win32/gdi/regions>

³⁶ <https://learn.microsoft.com/en-us/windows/win32/winstation/desktops>

³⁷ <https://learn.microsoft.com/en-us/windows/win32/procthread/job-objects>

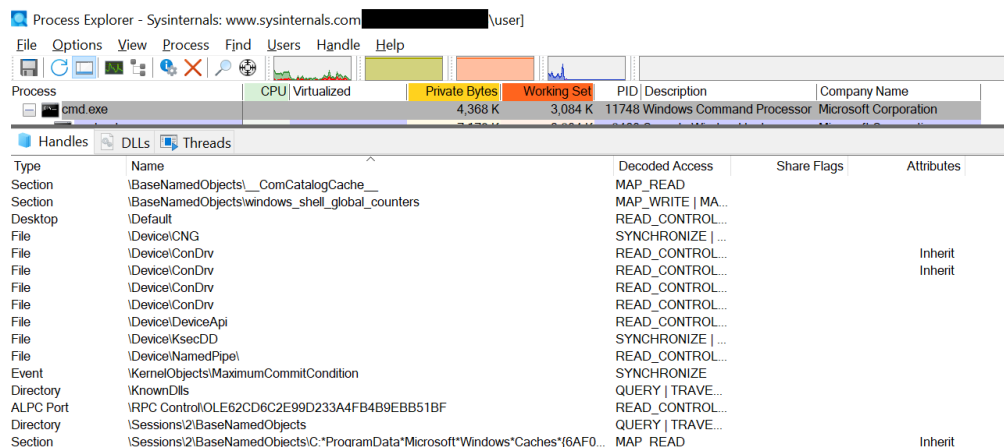
Object Manager

As part of the Windows operating system there is a single “Object Manager” that maintains all the objects³⁸. Among the tasks the “Object Manager” does are: creating objects, verifying that a process has a right to use an object, creating object handles³⁹ and returning them to the caller, maintaining resource quotas, duplicating handles and closing handles⁴⁰.

Overall, Windows has more than 25 types of objects. Some examples are: files, devices, threads, processes, events, mutexes, jobs, registry keys, sections, access tokens - as shown in the screenshot below taken from “Process Explorer”. The kernel routines which provide a direct interface with the “Object Manager” are prefixed with “Ob”⁴¹. Some examples are: “ObGetObjectSecurity”⁴² and “ObReferenceObjectByHandle”⁴³.

Moreover, we can go over the reference implementation of the “Object Manager” as part of ReactOS⁴⁴. There is also the internal header file⁴⁵.

Lastly, we can summarize the “Object Manager” as being responsible for keeping track of all the resources in Windows. It also provides a way for applications to access and manage those resources in a secure and efficient way⁴⁶.



³⁸ <https://medium.com/@boutnaru/windows-objects-2c289da600bf>

³⁹ <https://medium.com/@boutnaru/windows-handles-594b36c39d2f>

⁴⁰ <https://learn.microsoft.com/en-us/windows/win32/sysinfo/object-manager>

⁴¹ <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-object-manager>

⁴² <https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-obgetobjectsecurity>

⁴³ <https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-obreferenceobjectbyhandle>

⁴⁴ <https://github.com/reactos/reactos/tree/master/ntoskrnl/ob>

⁴⁵ <https://github.com/reactos/reactos/blob/master/ntoskrnl/include/internal/ob.h>

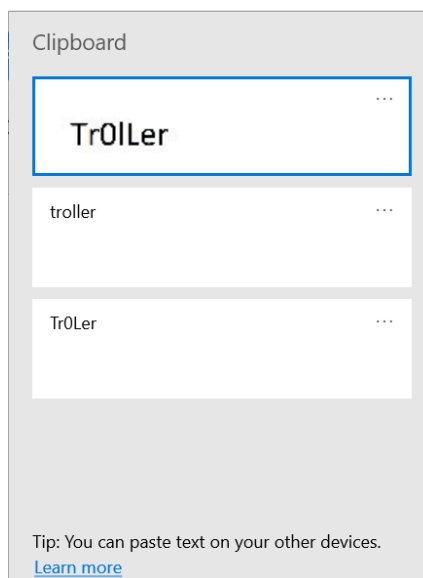
⁴⁶ <https://www.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Lectures/01-ObjectManager/ObjectManager.pdf>

Clipboard

Overall, the clipboard is a temporary storage in Windows that can store different data types. Among those data types are images and text. The operation which stores data in the clipboard is called “copy” (Ctrl+C), also we can use “cut” using “Ctrl+X”. In order to paste data from the clipboard we use “Ctrl+V”⁴⁷.

In Windows 10 we can store in the clipboard up to 25 items. We can access the “Clipboard History” using the keyboard shortcut “Winkey + V”. While pressing this shortcut a menu showing the current clipboard history - as shown in the screenshot below⁴⁸. We can see that the first item is an image and the other two are strings.

Lastly, in Windows 10/11 we can also copy text/images from one PC to another using a cloud based clipboard. We can also pin items that we would like to use. In order to do that we need to sign with a Microsoft account/work account⁴⁹.



⁴⁷ <https://www.howtogeek.com/671222/how-to-enable-and-use-clipboard-history-on-windows-10/>

⁴⁸ <https://www.emailoverloadsolutions.com/blog/windows-clipboard-history-feature>

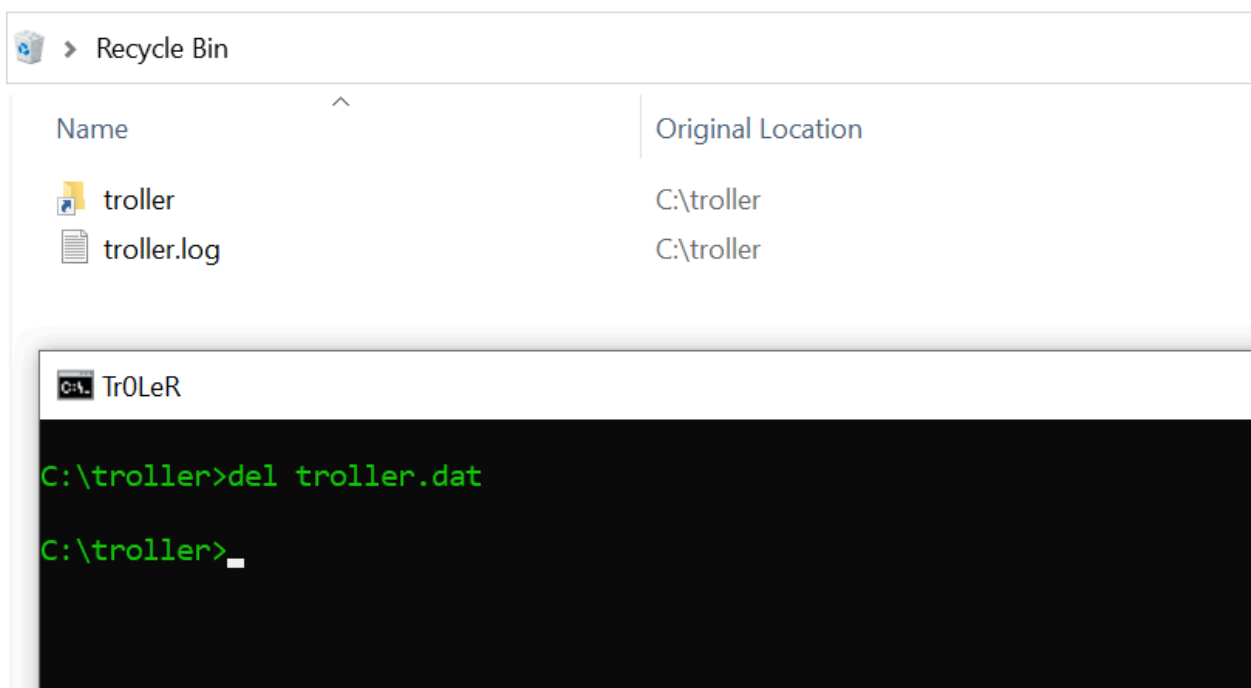
⁴⁹ <https://support.microsoft.com/en-us/windows/clipboard-in-windows-c436501e-985d-1c8d-97ea-fe46dd338c6>

Recycle Bin

The goal of the “Recycle Bin” is to provide a second chance for recovering files/directories that have been deleted. There is a nice trick to get into the “Recycle Bin” by using “cmd /c shell:RecycleBinFolder”. Of course we can just find a shortcut for it on the desktop⁵⁰.

Thus, we can say “Recycle Bin” is a folder where deleted items are stored temporarily. We can use the “Recycle Bin” to restore the files to their original location. However, it was not created in a manner in which we can use the files directly from the “Recycle Bin”⁵¹.

Moreover, files are moved to the “Recycle Bin” only if we delete them for “File Explorer” and not when doing it directly from other applications like “cmd.exe”⁵² - as shown in the screenshot below. By the way, holding the “Shift” key while deleting a file in “File Explorer” causes it not to be moved to the “Recycle Bin”. Lastly, we can go over a reference implementation of the “Recycle Bin” as part of ReactOS⁵³.



⁵⁰ <https://www.digitalcitizen.life/where-is-recycle-bin/>

⁵¹ <https://www.techopedia.com/definition/4675/recycle-bin>

⁵² <https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b>

⁵³ <https://github.com/reactos/reactos/tree/master/dll/win32/shell32/shellrecyclebin>

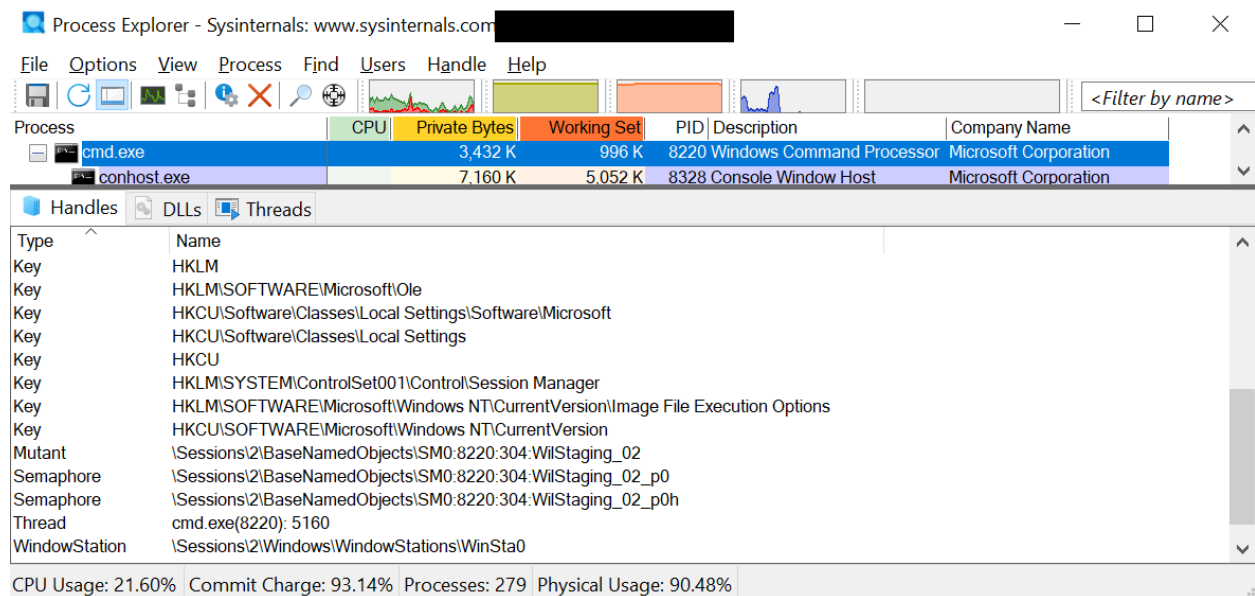
Handles

Overall, applications running in user mode can't directly access the Windows objects⁵⁴ or object data held by the "Object Manager". Due to that, an application needs to obtain a handle to the specific object. For each handle there is an entry in a handle table which resides in kernel space, there is one for each process⁵⁵.

Thus, an object in Windows is accessed by the user using a per process "handle table". Opening an object results in adding a pointer to the object to the specific "handle table". The return value is "an index" to that table⁵⁶.

We can see when using different Win32 API function like when opening/creating a file using "CreateFileW" the return value is of type HANDLE⁵⁷.

Lastly, we can use Sysinternals' tools like "Process Explorer" and handle.exe⁵⁸ in order to see which open handles each process has. An example showing the handles for "cmd.exe" using "Process Explorer" is shown in the screenshot below.



⁵⁴ <https://medium.com/@boutnaru/windows-objects-2c289da600bf>

⁵⁵ <https://learn.microsoft.com/en-us/windows/win32/sysinfo/handles-and-objects>

⁵⁶ https://www.cs.miami.edu/home/burt/journal/NT/handle_table.html

⁵⁷ <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew>

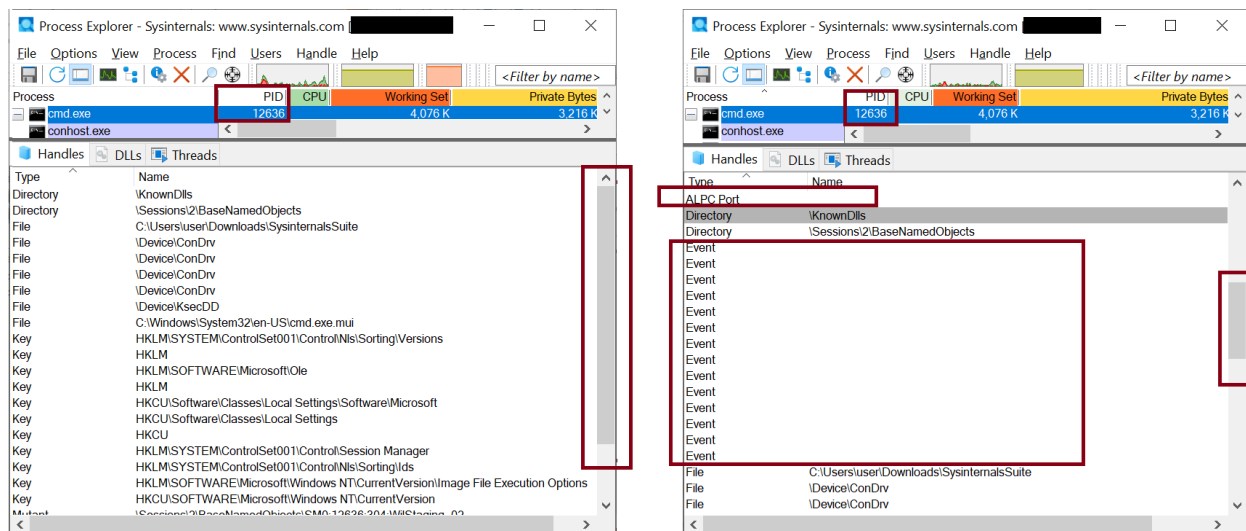
⁵⁸ <https://learn.microsoft.com/en-us/sysinternals/downloads/handle>

Unnamed Handles

Windows objects⁵⁹ are inaccessible directly from user-mode code. Handles are used as “indexes” in a per-process handle table (stored in the kernel address space). Thus, when opening an object a new handle entry is created⁶⁰. We are going to focus on handles which don’t have a name aka “unnamed handles” (as opposed to named handles).

Overall, we can’t treat an unnamed handle as a secure object by default. If an unnamed handle has a NULL DACL as part of its security descriptor⁶¹ it grants access to everyone. However, even if there is no name to a handle it does not mean it can’t be accessed - it just means we can’t access it using a name. We can still use the “DuplicateHandle” Win32 API function call to duplicate it⁶² or inherit the handle⁶³.

Lastly, we can see unnamed handles using “Process Explorer”. However, it is not enabled by default. In order to enable it we can press “View->Show Unnamed Handles and Mappings”. Then in the handles tab both named and unnamed handles are displayed - as shown in the screenshot below.



⁵⁹ <https://medium.com/@boutnaru/windows-objects-2c289da600bf>

⁶⁰ <https://medium.com/@boutnaru/windows-handles-594b36c39d2f>

⁶¹ <https://medium.com/@boutnaru/the-windows-security-journey-dacl-discretionary-access-control-list-c74545e472ec>

⁶² <https://learn.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-duplicatehandle>

⁶³ <https://devblogs.microsoft.com/oldnewthing/20150604-00/?p=45451>

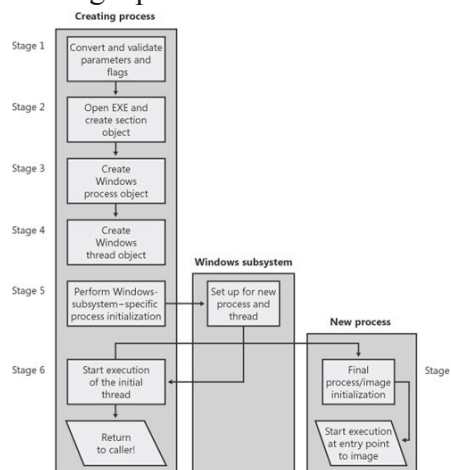
Processes

From the perspective of the Windows operating system an application is composed of one or more processes. Thus, we can say a process is an executing program⁶⁴. Moreover, every process provides the resources needed for the application to run. In order to support that a process in Windows has: a virtual memory address space, executable code, open handle table, unique identifier (PID), environment variables, priority, access token and more attributes⁶⁵.

Overall, for creating new processes under Windows we can use various Win32 API calls such as: “CreateProcessA”\“CreateProcessW”⁶⁶, “CreateProcessAsUserA”\“CreateProcessAsUserW”⁶⁷ and “CreateProcessWithLogonW”⁶⁸.

The function “CreateProcessA”\”CreateProcessW” creates the new process with the security context of the calling process. In the case of “CreateProcessAsUserA”\”CreateProcessAsUserW” we can provide an handle to the token⁶⁹ we want to grant the new process created. With “CreateProcessWithLogonW”, we can provide the username and password for the security context we want to execute the new process to execute with. For reference implementation I suggest going over ReactOS’s source code of “NtCreateProcessEx” which calls “PspCreateProcess”⁷⁰.

Lastly, a process has at least one execution flow which is a thread (aka the main/primary thread) - more on threads in a separate writeup. Also, I am going to describe the different data structures used by Windows in order to manage processes in future writeups. An overview of the stages Windows follows when creating a process is shown in the diagram below⁷¹.



⁶⁴ <https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads>

⁶⁵ <https://learn.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads>

⁶⁶ <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessasuserw>

⁶⁷ <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessasuserw>

⁶⁸ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw>

⁶⁹ <https://medium.com/@boutnaru/windows-security-access-token-81cd0000c64>

⁷⁰ <https://github.com/reactos/reactos/blob/master/ntoskrnl/ps/process.c#L1344>

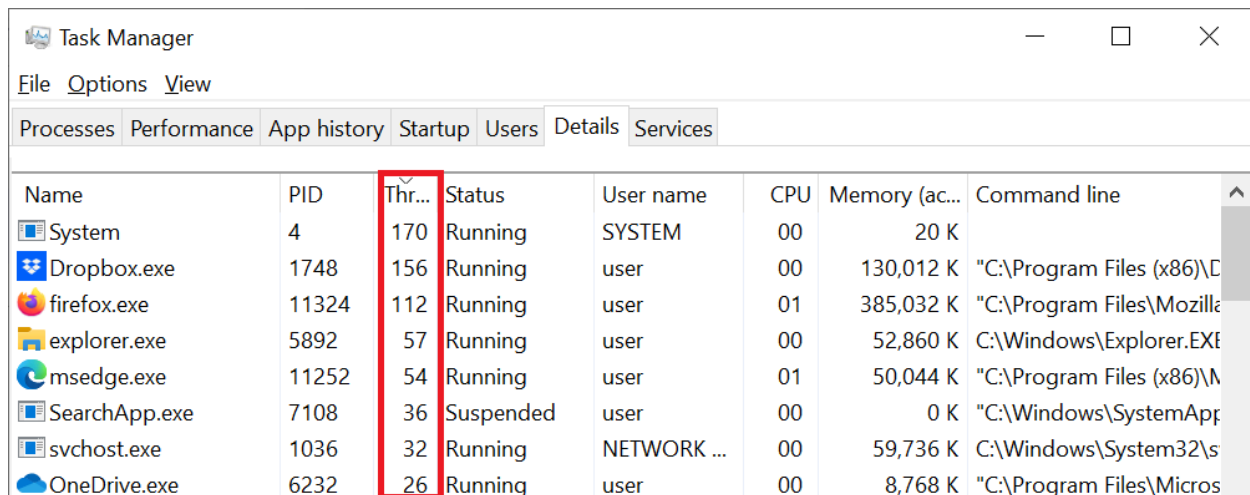
⁷¹ <https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=3>

Threads

In general a Windows process⁷² is composed of one or more threads. For the eyes of the operating system, a thread is the basic unit to which CPU time is allocated. A thread can run any code of the process, also the same code path can be executed in parallel by different threads⁷³. Each thread has its own identifier (TID, which stands for “Thread ID”). Also, every Windows process has at least one thread - as shown in the screenshot below.

Moreover, in order to create a thread within the virtual memory address space of the calling process we can use the “CreateThread” API call⁷⁴. If we want to create a thread in the virtual address space of a different process we can use “CreateRemoteThread”⁷⁵.

Lastly, each thread has its own stack area (both in user mode and in kernel mode) so each one of them gets its own execution flow. The default stack size in user-mode is 1MB⁷⁶. By the way, TID is also divisible by 4 like PID⁷⁷.



Name	PID	Thr...	Status	User name	CPU	Memory (ac...	Command line
System	4	170	Running	SYSTEM	00	20 K	
Dropbox.exe	1748	156	Running	user	00	130,012 K	"C:\Program Files (x86)\D
firefox.exe	11324	112	Running	user	01	385,032 K	"C:\Program Files\Mozilla
explorer.exe	5892	57	Running	user	00	52,860 K	C:\Windows\Explorer.EXE
msedge.exe	11252	54	Running	user	01	50,044 K	"C:\Program Files (x86)\M
SearchApp.exe	7108	36	Suspended	user	00	0 K	"C:\Windows\SystemApp
svchost.exe	1036	32	Running	NETWORK ...	00	59,736 K	C:\Windows\System32\s
OneDrive.exe	6232	26	Running	user	00	8,768 K	"C:\Program Files\Micros

⁷² <https://medium.com/@boutnaru/windows-process-923b9332c12>

⁷³ <https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads>

⁷⁴ <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread>

⁷⁵ <https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread>

⁷⁶ <https://learn.microsoft.com/en-us/windows/win32/procthread/thread-stack-size>

⁷⁷ <https://medium.com/@boutnaru/windows-why-pids-tids-are-4-divisible-6af16cf4621d>

Fiber

Fiber is a unit of execution which is not scheduled by the kernel of Windows, thus it needs to be scheduled by the application itself. Due to that, each fiber executes in the context of the thread⁷⁸ that had scheduled it. A code snippet for using fibers as shown in the image below⁷⁹. Overall, we can say that fibers are “lightweight threads” of execution. The big difference between them and OS threads is that they’re cooperatively scheduled as opposed to preemptively scheduled. This basically means that fibers yield themselves (they withdraw their execution) to allow another fiber to run. Sometimes you can find fibers called: “green threads”, “tasklets”, “coroutines”, “user-space threads” or “microthreads”⁸⁰.

Moreover, in order to use fibers we can leverage the Win32 API. “CreateFiber” allows the creation of a new fiber for a thread, the number of fibers per process is limited by the virtual memory⁸¹. We can use “ConvertThreadToFiber” for converting the current thread into a fiber, which is needed before scheduling other fibers⁸². For both of those function there is an extended version “CreateFiberEx”⁸³ and “ConvertThreadToFiberEx”⁸⁴. Lastly, for scheduling a fiber we can use the “SwitchToFiber” function⁸⁵. To retrieve the data associated with the current fiber we can use the “GetFiberData” function⁸⁶. When the fiber is not needed anymore just call “DeleteFiber”⁸⁷.

```
#include <stdio.h>
#include <windows.h>

// fiber function
void fiber_function(void* lpParam)
{
    // Print a message
    printf("Fiber created\n");
    printf("For educational purposes only\n");
    // Converting back into the main thread as fiber will not return to the main thread by itself
    SwitchToFiber(lpParam);
}

// main function
int main()
{
    // Converting thread to fiber
    LPVOID Context = ConvertThreadToFiber(NULL);
    // Creating fiber
    LPVOID lpFiber = CreateFiber(0, (LPFIBER_START_ROUTINE)fiber_function, Context);
    // Switching to fiber (executing fiber function)
    SwitchToFiber(lpFiber);
    // Printing a message
    printf("Fiber returned\n");
    // Deleting fiber
    DeleteFiber(lpFiber);
    // Return success
    printf("Exiting...\n");
    return 0;
}
```

⁷⁸ <https://medium.com/@bournaru/windows-threads-3a839fa67ae3>

⁷⁹ <https://de-engineer.github.io/Processes-threads-jobs-fibers/#creating-a-fiber>

⁸⁰ <https://graphitemaster.github.io/fibers/>

⁸¹ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfiber>

⁸² <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-convertthreadtofiber>

⁸³ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfiberex>

⁸⁴ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-convertthreadtofiberex>

⁸⁵ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-switchtofiber>

⁸⁶ <https://learn.microsoft.com/en-us/windows/win32/api/winnt/nf-winnt-getfiberdata>

⁸⁷ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-deletefiber>

Mailslot

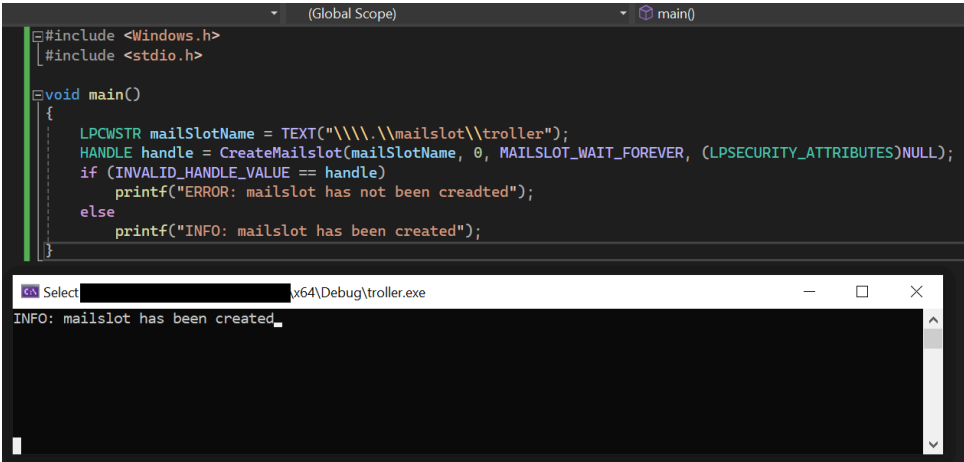
Mailslot is a Windows IPC⁸⁸ mechanism. It is used for one-way communication between endpoints. Thus, applications can store data in a mailslot. Those messages can be sent locally or over the network. The owner of the mailslot can retrieve messages stored in the mailslot⁸⁹.

Overall, we can think about a mailslot as a “pseudo file” that resides in memory (when the mailslot handle is closed that data is deleted). Due to that we can access it using the files API (like ReadFile/WriteFile/CreateFile). The data stored in a mailslot can be in any format, however it can be larger than 424 bytes when sent between computers⁹⁰.

Moreover, when creating a mailslot (by a mailslot server) its name must be in the following pattern “\\.\mailslot\[path\]name”. A mailslot client can write messages locally or remotely, when writing data to a remote mailslot we need to use the following “\\ComputerName\mailslot\[path\]name”. There is also a way for accessing every mailslot in a domain using “\\DomainName\mailslot\[path\]name” or “*\mailslot\[path\]name” in the system’s primary domain⁹¹.

In addition to that, for creating a mailslot we can use the API function “CreateMailslotA”⁹² or the wide character version of the API “CreateMailslotW”⁹³ - as shown in the screenshot below (this is just a toy example, please don’t use it “as is” in production code due to security reasons).

Lastly, the remote capability of mailslots is baked on the SMB (Server Message Block) protocol⁹⁴. By the way, Microsoft intends to remove the support of remote mailslots⁹⁵.



```
#include <Windows.h>
#include <stdio.h>

void main()
{
    LPCWSTR mailSlotName = TEXT("\\\\.\\mailslot\\troller");
    HANDLE handle = CreateMailslot(mailSlotName, 0, MAILSLT_WAIT_FOREVER, (LPSECURITY_ATTRIBUTES)NULL);
    if (INVALID_HANDLE_VALUE == handle)
        printf("ERROR: mailslot has not been created");
    else
        printf("INFO: mailslot has been created");
}
```

Output: INFO: mailslot has been created

⁸⁸ <https://medium.com/@boutnaru/windows-ipc-inter-process-communication-introduction-434c9287279b>

⁸⁹ <https://learn.microsoft.com/en-us/windows/win32/ipc/mailslots>

⁹⁰ <https://learn.microsoft.com/en-us/windows/win32/ipc/about-mailslots>

⁹¹ <https://learn.microsoft.com/en-us/windows/win32/ipc/mailslot-names>

⁹² <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createmailslot>

⁹³ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createmailslotw>

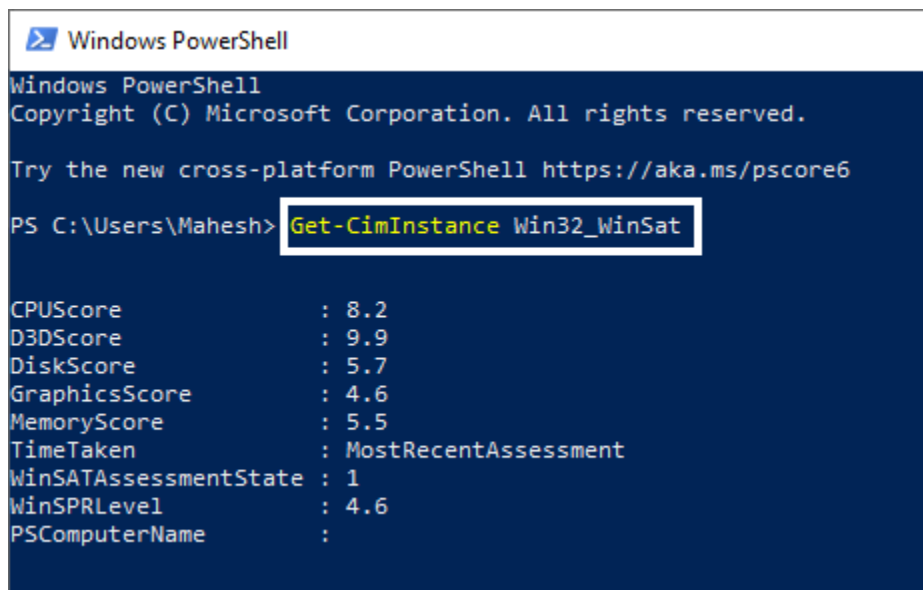
⁹⁴ <https://wiki.wireshark.org/Mailslot.md>

⁹⁵ <https://www.anoopnair.com/deprecation-of-remote-mailslots-in-windows/>

Windows Experience Index (WEI)

Windows Experience Index (WEI) is a scoring mechanism which allows us to understand how fast our computer is and which hardware shortcomings it has. WEI was first introduced as part of Windows Vista. The value of WEI can range from 1.0 (worst performance) to 9.9 (highest performance). It is used by WinSAT (Windows System Assessment Tool), which is based on the lowest score of : RAM, CPU, graphic chipset (or GPU) and hard disk⁹⁶.

Overall, for Windows 7 and 8 there was a GUI interface for viewing the WEI score (as part of the control panel). However, since Windows 10 it was removed, we can still calculate the score using “winsat.exe”⁹⁷. We can also use the WMI (Windows Management Instrumentation) class “Win32_WinSAT”, which summarizes the information of the recent assessment⁹⁸ - as shown in the screenshot below⁹⁹.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Mahesh> Get-CimInstance Win32_WinSat

CPU Score           : 8.2
D3D Score           : 9.9
Disk Score          : 5.7
Graphics Score      : 4.6
Memory Score        : 5.5
Time Taken          : MostRecentAssessment
WinSAT Assessment State : 1
WinSPR Level        : 4.6
PS Computer Name    :
```

⁹⁶ <https://winbuzzer.com/2020/03/07/windows-10-how-to-get-the-windows-experience-index-wei-score-xcxwb/>

⁹⁷ <https://webmarks.info/howto/>

⁹⁸ <https://learn.microsoft.com/en-us/windows/win32/winsat/win32-winsat>

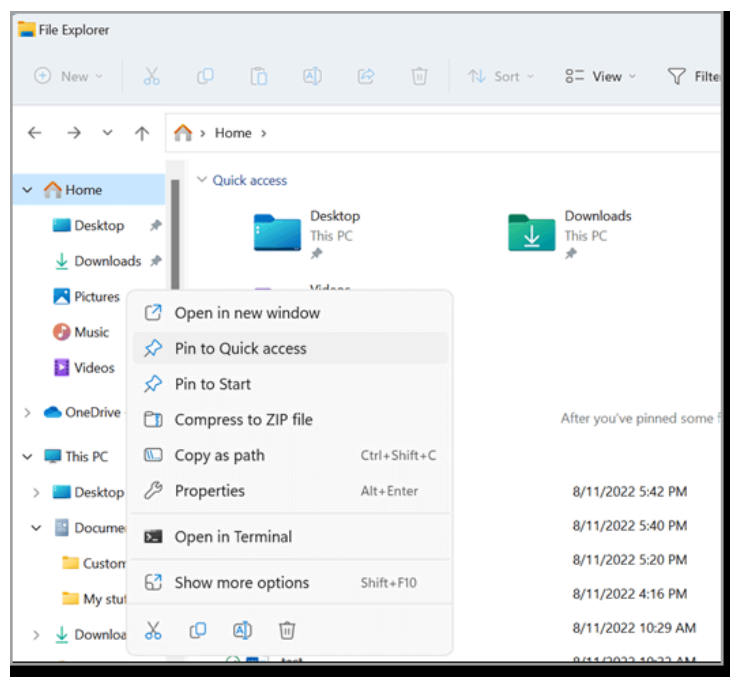
⁹⁹ <https://techtipsinfinite.blogspot.com/2020/05/blog-post.html>

File Explorer (previously Windows Explorer)

By using “File Explorer” users can access/manipulate files very quickly - as shown in the screenshot below. We can open the “File Explorer” using the shortcut “WinKey+E”¹⁰⁰.

Also, since Windows 10 “Windows Explorer” name was changed to “File Explorer”¹⁰¹. Moreover, it is part of the graphical shell which is started when a user logs on to the system - “explorer.exe”¹⁰². We can think about it like “/bin/bash” in Linux but in this case it’s a graphical shell.

Lastly, “Windows Explorer” can be extended (Windows Shell Extensions), this is based on COM¹⁰³ objects. The shell extensions can be: toolbars, shell extensions handlers or a namespace extensions which allow certain folders to be displayed differently - check out “%windir%\Fonts” as an example¹⁰⁴.



¹⁰⁰ <https://support.microsoft.com/en-us/windows/find-and-open-file-explorer-ef370130-1cca-9dc5-e0df-2f7416fe1cb1>

¹⁰¹ <https://support.microsoft.com/en-us/windows/windows-explorer-has-a-new-name-c95f0e92-b1aa-76da-b994-36a7c7c413d7>

¹⁰² <https://medium.com/@boutnaru/the-windows-process-journey-explorer-exe-windows-explorer-9a96bc79e183>

¹⁰³ <https://medium.com/@boutnaru/windows-com-component-object-model-71a76a97435c>

¹⁰⁴ https://en.wikipedia.org/wiki/File_Explorer

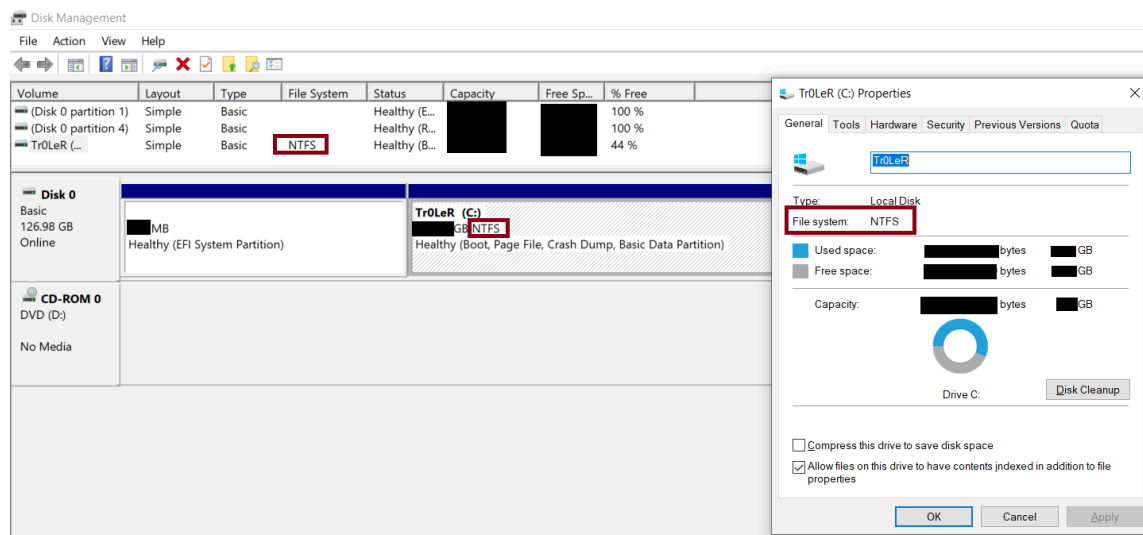
NTFS (New Technology File System)

NTFS (New Technology File System) is the primary file system used by Windows Workstation/Server versions in the last 30 years - as shown in the screenshot below. It was introduced as part of Windows NT 3.1 (1993) as a proprietary journaling file system by Microsoft¹⁰⁵.

Overall, it provides different features: increased reliability, increased security, support for large volumes, support for long file names and extended path names and flexible allocation of capacity, journaling, compression, hardlinks, volume shadow copy, transactions and quotas. I am going to detail only part of those features, for more information you can read the references¹⁰⁶.

Increased security is implemented by NTFS support file/folder permissions in using ACLs¹⁰⁷: both DACLs¹⁰⁸ and SACLs¹⁰⁹. Also, there is support for file encryption using EFS (Encrypting File System) or a part of disk encryption using Bitlocker.

Increased reliability is implemented by NTFS using log file and checkpoint information to restore consistency. Since Windows Server 2008 there is also a feature called “Self-Healing NTFS”, which attempts to correct corruptions of the file system online. This is done without the need for “chkdsk.exe”¹¹⁰. Lastly, we can also check out the reference implementation for NTFS from ReactOS¹¹¹.



¹⁰⁵ <https://en.wikipedia.org/wiki/NTFS>

¹⁰⁶ <https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview>

¹⁰⁷ <https://medium.com/@boutnaru/the-windows-security-journey-acl-access-control-list-b7d9a6fe4282>

¹⁰⁸ <https://medium.com/@boutnaru/the-windows-security-journey-dacl-discretionary-access-control-list-c74545e472ec>

¹⁰⁹ <https://medium.com/@boutnaru/the-windows-security-journey-sacl-system-access-control-list-32488dcc80d7>

¹¹⁰ [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc771388\(v=ws.10\)](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc771388(v=ws.10))

¹¹¹ <https://github.com/reactos/reactos/tree/master/drivers/filesystems/ntfs>

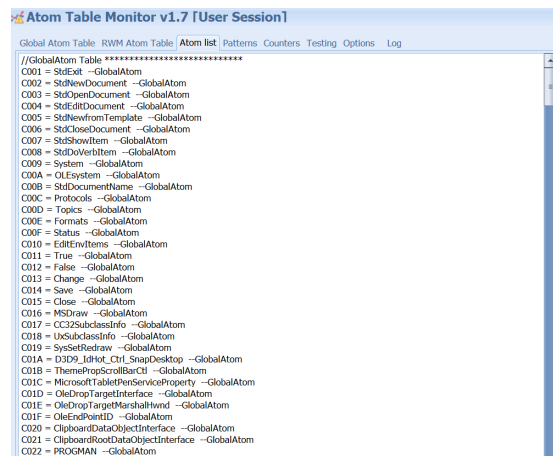
Atom Table

An “Atom Table” is a system-defined table which is used to store a string that has a specific identifier. When a program stores a string in an atom table it gets a 16 bit integer number (“atom”) which can be used to access the string (“atom name”). There is not one atom table¹¹².

Overall, there are two types of atom tables: “Global Atom Table” and “Local Atom Table”. The global table is defined for all processes on the system while the local one is private for the process which created it. By the way, there are two types of atoms: “String/Integer Atoms”¹¹³.

In regards to “String Atoms”, they have specific properties. They can’t be longer than 255 bytes which can be returned within the range “0xC000-0xFFFF”. Also, the reference count is incremented when the string is added and decremented when removed. “Integer Atoms” are in the range of “0x0001-0xBFFF”, they don’t have any reference count or storage limitation¹¹⁴.

Moreover, we can use different Win32 API calls to manipulate an “Atom String”. In regards of the global table we can use the following functions to add and find them: “GlobalAddAtomA”/”GlobalAddAtomW”¹¹⁵ and “GlobalFindAtomA”/”GlobalFindAtomW”¹¹⁶. In regards to the local table we can use the following functions: “AddAtomA”/”AddAtomW”¹¹⁷ and “FindAtomA”/”FindAtomW”¹¹⁸. Lastly, we can use the “atom-table-monitor”¹¹⁹ for viewing the global atom table - as shown in the screenshot below. Also, we can go over a reference implementation of “Atom Management”¹²⁰ and “Executive Atom Functions”¹²¹ as part of ReactOS.



¹¹² <https://learn.microsoft.com/en-us/windows/win32/dataxchg/about-atom-tables>

¹¹³ https://www.oreilly.com/library/view/microsoft-windows-2000/0672319330/0672319330_ch23lev1sec1.html

¹¹⁴ <https://bsodtutorials.wordpress.com/2015/11/11/understanding-atom-tables/>

¹¹⁵ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-globaladdatomw>

¹¹⁶ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-globalfindatomw>

¹¹⁷ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-addatomw>

¹¹⁸ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-findatomw>

¹¹⁹ <https://github.com/JordiCorbilla/atom-table-monitor/>

¹²⁰ <https://github.com/reactos/reactos/blob/master/sdk/lib/rtl/atom.c>

¹²¹ <https://github.com/reactos/reactos/blob/master/ntoskrnl/ex/atom.c>