Windows Dependencies Help Contents

Version 1.0.0.2505 © WinDepends project

What is Windows Dependencies (WinDepends) for?

Windows Dependencies a.k.a. **WinDepends** is a free utility that scans any 32-bit or 64-bit Windows module (exe, dll, ocx, sys, etc.) and builds a hierarchical tree diagram of all dependent modules. For each module found, it lists all the functions that are exported by that module, and which of those functions are actually being called by other modules. Another view displays the minimum set of required files, along with detailed information about each file including a full path to the file, base address, version numbers, machine type, debug information, and more.

WinDepends is also maybe useful for troubleshooting system errors related to loading and executing modules. WinDepends detects some of the common application problems such as missing modules, invalid modules, import/export mismatches, mismatched machine types of modules. It runs on Windows 10 and 11. It can process any 32-bit or 64-bit Windows module of a valid Portable Executable format. It handles the following types of module dependencies: implicit, forwarded and delay-loaded. A detailed help is included.

WinDepends is heavily inspired by Dependency Walker, developed by Steve P. Miller © Microsoft Corp.

WinDepends is completely free to use and fully open-source under the following MIT license:

Back to main page

Frequently Asked Questions (FAQ)

Q: WinDepends seems to only show some of my application's dependencies. Why doesn't it show all of them?

A: When you first open a module in **WinDepends**, it only shows implicit, forwarded, and delay-load dependencies. Many dependencies are loaded dynamically and will not be detected until you profile the application with help of other tool like for example Sysinternals Process Monitor. For more information, see Types of Dependencies Handled by WinDepends.

Q: Can WinDepends help me figure out why my component won't register? [or] Why does REGSVR32.EXE fail to register my DLL, but WinDepends does not show any error with my DLL?

A: Many modules need to be "registered" on a computer before they will work. This includes most ActiveX controls, OCXs, COM components, ATL components, Visual Basic components, and many others. These types of modules are usually registered with REGSVR32.EXE or something similar. For the most part, REGSVR32.EXE loads your DLL, calls **GetProcAddress** for the DLL's **DllRegisterServer** function, then calls that function. A common failure is when your DLL relies on another DLL that is missing or not registered. If you just open your DLL in WinDepends, you may or may not see a problem, depending on the type of registration failure.

Q: How do I view the parameter and return types of a function?

A: For most functions, this information is simply not present in the module. The Windows' module file format only provides a single text string to identify each function. There is no structured way to list the number of parameters, the parameter types, or the return type. However, some languages do something called function "decoration" or "mangling", which is the process of encoding information into the text string. For example, a function like **int Foo(int, int)** encoded with simple decoration might be exported as **_Foo@8**. The 8 refers to the number of bytes used by the parameters. If C++ decoration is used, the function would be exported as **?Foo@@YGHHH@Z**, which can be directly decoded back to the function's original prototype: **int Foo(int, int)**. WinDepends supports C++ undecoration by using the Undecorate C++ Functions Command.

Back to main page

Q: Why are my function names exported differently then I declare them?

A: Many compilers "decorate" function names by default. Unless you give the compiler specific instructions on how to export functions, a function like **int Foo(int, int)** may end up getting exported as **_Foo@8**, or even **?Foo@@YGHHH@Z** if C++ decoration is used. Languages like C++ allow function overloading, which is the ability to declare multiple functions with the same name, but with different parameters. Because of this, each function must have a unique signature string since exporting just the name would cause a name conflict. To disable C++ decoration, you can use the **extern "C"** notation when declaring your functions in a C++ source file. To prevent decoration altogether, you can add a DEF file to your C/C++ project and declare the actual function names you want exported.

Q: Why do some modules show up more than once under a single parent module?

A: **WinDepends** may show a module more than once to inform you that it is a dependency for more than one reason. It is possible for a module to show up as an implicitly linked dependency, a forwarded dependency, or be an API set contract which resolves to the same module name multiple times, all under a single parent module. See the Module Dependency Tree View for more details. In reality, only one copy of the module resides in memory during run-time.

Q: Does WinDepends handle API set contracts, and what are they?

A: Yes, **WinDepends** handles Windows API set contracts. API sets are a special mechanism for DLL redirection introduced in Windows 7. All versions of Windows share a common base of operating system components called the core OS (referred to as *OneCore* in some contexts). Within core OS components, Win32 APIs are organized into functional groups called API sets. The purpose of API sets is to provide architectural separation between the host DLL in which a given Win32 API is implemented and the functional contract to which the API belongs. **WinDepends** handles API sets at the native level and can parse various versions of API set schemas, such as V2 (Windows 7), V4 (Windows 8/8.1), and V6 (Windows 10/11). If configured WinDepends can highlight API set contract dlls in the Module Dependency Tree View.

Q: Will WinDepends work with COM, Visual Basic, or .NET modules?

A: Yes. **WinDepends** will work with any 32-bit or 64-bit Windows module, regardless of what language was used to develop it. However, many languages have their own way to specify dependency relationships between modules. For example, COM modules may have embedded type libraries and registration information in the registry, and .NET modules may use .NET assemblies. These techniques are all implemented as layers above the core Windows API. In the end, these layers still need to call down to the core Windows functions like **LoadLibrary** and **GetProcAddress** to do the actual work. It is at this core level that **WinDepends** understands what is going on. So, while **WinDepends** may not understand all the language specific complexities of your application, it will still be able to track all module activity at a core Windows API level.

Q: What do all the version numbers mean?

A: See the Overview of Module Version Numbers section for the details.

Q: How can I send the results of a session to someone?

A: **WinDepends** supports several ways to capture the data in a session. All the views support simple copying from them using the Copy Command. It also supports method of saving the entire session to a file. You can save the results to a **WinDepends** Session view (WDS) file, which can be loaded by **WinDepends** on another computer to see the captured results from your computer. For more information on saving the session to a file, see the Save Command.

Q: What do all the icons mean?

A: Each view in **WinDepends** has detailed help describing what the icons mean for that view. See the Module Session View section for a list of views.

Q: WinDepends open dialog is not showing a file that I want to open. How can I fix this?

A: Windows may "hide" certain system files (like DLLs) from the user. You have to configure system settings depending on current Windows version.

On Windows 11: Open File Explorer from the taskbar. Select View > Show > Hidden items.

On Windows 10: Open File Explorer from the taskbar. Select View > Options > Change folder and search options. Select the View tab and, in Advanced settings, select Show hidden files, folders, and driver and press OK.

Q: How do I uninstall WinDepends?

Back to main page

A: **WinDepends** does not have a setup or uninstall program. It was designed to simply run when you want it, and delete if you don't need it anymore. If you have told **WinDepends** to handle certain file extensions, you will probably want to remove those associations before deleting the program. This can be done by using the Configuration > Shell Integration command. The files to delete when WinDepends is no longer needed are **WinDepends.exe**, **WinDepends.Core.x64.exe**, **WinDepends.Core.x86.exe** and **WinDepends.pdf**. You may also want to remove configuration file which is named **WinDepends.x64.settings.bin** or **WinDepends.x86.settings.bin** depending on the program architecture. Sometimes WinDepends maybe also distributed with debug pdb files usually having same names as program components but “.pdb” extension. They also can be safely removed.

Back to main page

Using WinDepends for General Information about Modules

WinDepends provides valuable information about the module layout of a particular application and details on each module:

- A complete module dependency tree diagram of all the modules required by a particular application.
- A list of all functions exported from each module. These lists include functions exported by name, functions exported by ordinal, and functions that are actually forwarded to other modules. Named C++ functions can be shown in their native decorated format, or can be expanded into human readable function prototypes including return types and parameters types.
- A list of functions that are actually called in each module by other modules. These lists can help developers understand why a particular module is being linked with an application, and also provides information on how to remove unneeded modules from being dependencies.
- A list of the minimum set of files that are required in order for a module to load and run. This list can be very useful when copying files to another computer or creating setup scripts.
- For each individual module found, the following information is provided…
    - Full path to the module file.
    - Date and time of the module file. If the module was built using reproducible builds then reproducible hash will be displayed instead.
    - Date and time the module was actually built.
    - Size of the module file.
    - Attributes of the module file.
    - The module checksum from when the module was built.
    - The actual module checksum.
    - Type of CPU that the module was built for.

Back to main page

- Type of subsystem that the module was built to run in.
- Type of debugging symbols that are associated with the module.
- The preferred base load address of the module.
- The virtual size of the module.
- The file version found in the module's version resource.
- The product version found in the module's version resource.
- The image version found in the module's file header.
- The version of the linker that was used to create the module file.
- The version of the OS that the module file was built to run on.
- The version of the subsystem that the module file was built to run in.
- A possible error message if any error occurred while processing the file.

Back to main page

Overview of Module Version Numbers

There are four version fields that every Windows module is guaranteed to have. They are all two-part version numbers (#.#). They include:

Image Version	This value is set by the developer of the module by using the VERSION statement in their DEF file or by using the /VERSION linker option. It usually represents the version of the module or product that the module is part of, but can contain any value since it is up to the developer to set it. If the developer does not specify a version, then this value will default to 0.0. This value may be used as a last resort when comparing two modules to check which module is newer.
OS Version	This value represents which version of the operating system the module was designed to run on. Certain functions may behave differently depending on this value in order to remain compatible with applications built for a particular operating system version.
Subsystem Version	This value represents which subsystem version the module was designed to run on. Most modules use the default value, but developers can override the default by using the /SUBSYSTEM linker option if they wish to target a particular subsystem version other than the default. Certain subsystem functions may behave differently depending on this value in order to remain compatible with applications built for a particular subsystem version.
Linker Version	This value represents the version of the linker that was used to build the module. It can be used to determine if a specific linker feature was available at the time the module was built. For example, delay-load dependencies is a new feature introduced with version 6.0 of the linker, so if this value is less than 6.0, the module shouldn't have any delay-load dependencies.

In addition to the four standard version values, developers can add four more optional version values by including a VERSION_INFO resource as part of their resource file. This resource structure has two four-part numeric fields (#.#.#.#) and two text fields. They include:

File Version Value	This field is known as the "FILEVERSION" field in the VERSION_INFO resource structure. This numerical value usually represents the version of the module itself, but can contain any value since it is up to the developer to set it. This is the value that most programs use when comparing two modules to check which module is newer.

Back to main page

Product Version Value	This field is known as the "PRODUCTVERSION" field in the VERSION_INFO resource structure. This numerical value usually represents the version of the product that this module is part of, but can contain any value since it is up to the developer to set it. For example, "Acme Tools version 7.0" is a set of ten utilities, including "Acme AntiVirus version 2.5". The virus checker executable might have a file version of 2.5.0.0 and a product version of 7.0.0.0
File Version Text	This field is known as the "FileVersion" field in the VERSION_INFO resource structure. This text string usually represents the version of the module itself, but can contain any text string since it is up to the developer to set it.
Product Version Text	This field is known as the "ProductVersion" field in the VERSION_INFO resource structure. This text string usually represents the version of the product that this module is part of, but can contain any text string since it is up to the developer to set it.

WinDepends shows the true FILEVERSION and PRODUCTVERSION version values and not the text string versions. Other applications, like the Windows Properties dialog, show the text string values since that is what the developer of the module wants the average non-technical user to see. For example, you may see only "2.0" in the Windows Properties dialog for a module when its real version is 2.0.5.2034. If you want to know the true version of a file, you should use **WinDepends** and not the Windows Properties dialog.

Back to main page

Types of Dependencies Handled by WinDepends

There are several ways a module can be a dependent of another module:

1. **Implicit Dependency** (also known as a load-time dependency or sometimes incorrectly referred to as static dependency): Module A is implicitly linked with a LIB file for Module B at compile/link time, and Module A's source code actually calls one or more functions in Module B. Module B is a load time dependency of Module A and will be loaded into memory regardless if Module A actually makes a call to Module B at run-time. Module B will be listed in Module A's import table.
2. **Delay-load Dependency**: Module A is delay-load linked with a LIB file for Module B at compile/link time, and Module A's source code actually calls one or more functions in Module B. Module B is a dynamic dependency and will only be loaded if Module A actually makes a call to Module B at run-time. Module B will be listed in Module A's delay-load import table.
3. **Forward Dependency**: Module A is linked with a LIB file for Module B at compile/link time, and Module A's source code actually calls one or more functions in Module B. One of the functions called in Module B is actually a forwarded function call to Module C. Module B and Module C are both dependencies of Module A, but only Module B will be listed in Module A's import table.
4. **Explicit Dependency** (also known as a dynamic or run-time dependency): Module A is not linked with Module B at compile/link time. At runtime, Module A dynamically loads Module B via a **LoadLibrary** type function. Module B becomes a run time dependency of Module A, but will not be listed in any of Module A's tables. This type of dependency is common with OCXs, COM objects, and Visual Basic applications.
5. **System Hook Dependency** (also known as an injected dependency): This type of dependency occurs when another application hooks a specific event (like a mouse event) in a process. When that process produces that event, the OS can inject a module into the process to handle the event. The module that is injected into the process is not really a dependent of any other module, but does resides in that process' address space.

Case 1, 2, and 3 can easily be detected by just opening a module in **WinDepends**. Case 4 and 5 require runtime profiling with system monitoring tools like for example Sysinternals Process Monitor.

Back to main page

The Module Session View

A module session view split into the following sub-views:

- Module Dependency Tree View
- Modules List View
- Parent Import Function List View
- Export Function List View
- Log View

All views support right-click context menus to commonly used commands for that view.

Back to main page

The Module Dependency Tree View

The Module Dependency Tree View displays a hierarchical view of all the modules' dependencies.

WinDepends starts with the root module you choose to open and scans its import tables to build a list of required dependent modules. It then scans each of these dependent modules for **their** dependencies. This recursion continues until all modules and their dependencies have been processed **or the maximum recursion depth is reached** (the depth limit is configurable in the program settings).

Duplicate modules are marked with a small arrow in the middle of their accompanying image (see below). To determine what the branch would have looked like if **WinDepends** had processed it, use the Highlight Original Instance Command to find the original instance of the module in the tree.

While processing the dependency tree, **WinDepends** performs several validity checks along the way. It checks to make sure each module is a valid 32-bit or 64-bit Windows module. It checks for mismatched binaries with different CPU architecture. It scans import and export function tables looking for unresolved external functions.

The Auto Expand setting controls how much of the tree is initially seen after loading a module. When this option is turned on, the entire tree will be displayed. When the option is turned off, only the root module, its immediate dependencies, and modules with errors will be shown.

Modules can be displayed using full file paths or just the file name to conserve screen space. You can control what is displayed using the Full Paths option. You may also copy the selected module's file name or path to the clipboard by selecting the Copy Command. The actual text copied will differ depending on how the Full Paths option is set.

The following is a table of the primary images that can accompany each module in the dependency tree. This list is just a subset of all the possible images. Actual images can be a combination of one or more of the following images:

Back to main page

Normal Images

Normal 32-bit module with no errors.

Duplicate module. This module has already been processed somewhere else in the tree. You can use the Highlight Original Instance Command to find the original instance of the module in the tree.

Forwarded module. This module is a dependency because the parent module has forwarded one of its functions to this module.

Delay-load module. This module will be dynamically loaded if any of its exported functions are actually called at run-time.

64-bit module. This module is designed to run on a 64-bit versions of Windows. Modules are assumed to be 32-bit if this image is not present.

Warning and Error Images

Missing module. This module could not be found in the search path. See the Configure Module Order Command for more information.

Invalid module. See the Log View for an error message describing the module error.

Module warning. This module is either missing one or more export functions that are required by its parent module, is of the wrong CPU type, or failed to initialize when being loaded. For a missing export, the Parent Import Function List View will list the actual unresolved functions that are causing the problem. For implicit dependencies, this is an error that will cause the parent module to fail to load. If the module failed to load or initialized, then check the Log View for details on the failure.

Delay-load module warning. This module is either missing one or more export functions that are required by its parent module, or is of the wrong CPU type. For a missing export, the Parent Import Function List View will list the actual unresolved functions that are missing. For delay-load dependencies, this is most likely not an error since one reason developers use delay-load modules is when they are unsure if a particular function exists in dependent module. Parents of delay-load modules have techniques for recovering from missing exports in the delay-loaded dependent module.

Back to main page

It is possible for a module to show up more than once as a dependency of a single parent module. **WinDepends** does this to inform you that this module is a dependency for more than one reason. A module can show up as an implicitly linked dependency, a forwarded dependency, and a dynamic dependency, all under a single parent module.

For example, if module A implicitly links to module B, you will see module B under module A as an implicit dependency. The functions listed in the Parent Import Function List View for that instance of module B are what is required for module A to be able to successfully load.

Back to main page

The Modules List View

The Module List View displays a list of all unique modules that are dependencies for the root module you opened. This list defines the set of files needed for the module to load and execute as a running process.

Modules can be displayed using full file paths or just the file name to conserve screen space. You can control what is displayed using the Full Paths option. You may also copy the selected modules' file names or paths to the clipboard by selecting the Copy Command. The actual text copied will differ depending on how the Full Paths option is set. If more than one module is selected, a list will be copied to the clipboard with carriage returns after each module.

There is not a one-to-one relationship between the modules listed in this list view and the modules listed in the Module Dependency Tree View. This list view shows the unique set of modules, where as the tree view shows all the module relationships. A module like KERNEL32.DLL may show up dozens of times in the tree view since many other modules depend on it, but it will only show up once in this list view. Some instances of KERNEL32.DLL might be implicitly loaded, while others may be dynamically loaded. Some might have import / export mismatch errors, while others may have no errors.

The Module List View contains several columns of information about each module. These columns include:

Image	The graphic icon displaying state of the module.
Module	Full path or file name for the module file. See the Full Paths option for toggling between the two modes. Additionally if module is an API set contract there will be API set contract file name or resolved module full path/filename, see Show Resolved API sets command for toggling between the two modes.
File Time Stamp	Date and time of the module file. This is the time that the file was last saved.

Back to main page

Link Time Stamp	Date and time that the module was built. This is a value that the linker store in the file itself. If the module was built using reproducible builds then reproducible hash will be displayed instead.
File Size	Size of the module file.
Attr.	Attributes of the module file. Possible values are R (read only), H (hidden), S (system), A (archive), C (compressed), T (temporary), O (offline), and E (encrypted).
Link Checksum	The module checksum from when the module was built. This value is set by using the linker's /RELEASE command line option. If this linker option is not specified, then the checksum may be zero. This value will be shown in red if it is not zero and does not match the actual module checksum. If the values do not match, it means that the module has been modified after it was built.
Real Checksum	The actual module checksum. This value is computed by WinDepends and should match the checksum computed by the linker when the module was built.
CPU	Type of CPU that the module was built for. Possible values are x86, x64, Intel64, ARM Thumb/Thumb-2, ARM Thumb-2, ARM64. This value will be shown in red if it does not match the CPU type of the root module in the session. This value is set by using the linker's /MACHINE command line option.
Subsystem	Type of subsystem that the module was built to run in. Possible values are GUI, Console, WinCE 1.x GUI, OS/2 console, Posix console, EFI Application, EFI Boot Driver, EFI Runtime Driver, EFI ROM, Xbox, BootApp.
Symbols	Type of debugging symbols that are associated with the module. Possible values are None, DBG (debug), PDB (program database), CV (codeview), COFF (common object file format), FPO (frame pointer omission), OMAP, and Borland.
Preferred Base	The preferred base load address of the module. This will be 32-bits for 32-bit modules and 64-bits for 64-bit modules. This value is set by using the linker's /BASE command line option.

Back to main page

Virtual Size	The virtual size of the module. This is the size of memory that will be reserved for the module to be mapped into.
File Ver	The file version found in the module's version resource. This value represents the FILEVERSION field in the VERSION_INFO resource structure. It will read "N/A" if the module does not contain a VERSION_INFO resource.
Product Ver	The product version found in the module's version resource. This value represents the PRODUCTVERSION field in the VERSION_INFO resource structure. It will read "N/A" if the module does not contain a VERSION_INFO resource.
Image Ver	The image version found in the module's file header. This value is set by using the linker's /VERSION command line option.
Linker Ver	The version of the linker that was used to create the module file.
OS Ver	The version of the OS that the module file was built to run on.
Subsystem Ver	The version of the subsystem that the module file was built to run in. This value is set by using the linker's /SUBSYSTEM command line option.

The module list can be sorted on the data in any column in the list. Simply click on the column header button for the column you wish to sort by. An arrow (^) is displayed in the column header for the column that the list is currently sorted by. You can also size a column to its "best fit" width by double-clicking the divider line between two columns in the column header. You can search for text in the currently sorted column by simply typing in the first few characters of the item you wish to find.

Back to main page

The Parent Import Function List View

The Parent Import Function List View displays the list of parent import functions for the currently selected module in the Module Dependency Tree View. Parent import functions are functions that are actually called in the given module by the parent module.

For implicit and forward dependencies, the selected module needs to export every function that the parent is importing from it. If the selected module does not export one of the functions that the parent module expects to call, then an unresolved external error will occur if the module is attempted to be loaded. See the Export Function List View for viewing the selected module's export functions.

WinDepends searches the exported function list for every parent import function to ensure there is a match. If any function is unresolved, then the function is marked with an error image (see below) and the module is mark with an error image as well in the Module Dependency Tree View and the Module List View.

The Parent Import Function List View can also help you locate unnecessary modules in an application. The fact that the parent module is calling functions in the selected module is what makes the selected module a dependency of the parent. As a developer, if you can safely stop the parent module from calling all the functions listed in the parent import function list for a given module, then that module will no longer be a dependent of the parent module.

C++ functions can be displayed in their native decorated format or in a human readable undecorated format. See the Undecorate C++ Functions Command for more information. You may also copy the selected function names to the clipboard by selecting the Copy Command. The actual text copied will differ depending on how the Undecorate C++ Functions option is set. If more than one function is selected, a list will be copied to the clipboard with carriage returns after each function.

Back to main page

The following are the primary images that can accompany each function in the parent import list:

C	Resolved C import.
C++	Resolved C++ import. C++ function can be viewed in their native decorated form or in a human readable undecorated form. See the Undecorate C++ Functions Command for more information.
O#	Resolved ordinal import.
C	Unresolved C function (similar images also exist for C++ and ordinal functions). This function is called by the parent module, but it is not exported from the current module. This is often referred to as an "unresolved external function". If this module is an implicit or forwarded dependency, then the parent module will fail to load. If this module is a delay-load dependency, then the parent module will most likely recover from the missing dependency, as that is a feature of using delay-load dependencies.

The Parent Import Function View is comprised of five columns:

Image	See the above list for descriptions. The header for this column has the letters "PI" in it, which just stands for "Parent Imports"
Ordinal	The ordinal value of the imported function, if the function is imported by ordinal. This value can be "N/A" if the function is imported by name.
Hint	The hint value for the imported function. The hint value is used internally by the operating system's loader to quickly match imports with exports. It is used as an index into the array of exported functions in the selected module.
Function	The name of the imported function, if the function is imported by name. This can be "N/A" if the function is imported by ordinal. C++ functions can be viewed in their native decorated form or in a human readable undecorated form. See the Undecorate C++ Functions Command for more information. You may also see "<invalid string>" as a function name, which means a call to GetProcAddress was made with an invalid string, or "<empty-string>", which means GetProcAddress was called with an empty string.

Back to main page

Entry Point	The entry point memory address for the function. For implicit and forward dependencies, this field often reads "Not Bound", which means that the entry point address will not be known until load time. If an address is given, then the parent module has been pre-bound by a program like BIND. Binding is the process of walking the import list of a module and the export list of all its dependent modules, in order to fill in the import list with the absolute addresses to the functions it references. This job is usually done by the loader as each module is loaded, but can be skipped if the modules have been pre-bound. Pre-binding is an optimization that calculates the absolute addresses based off of the modules' preferred base addresses and stores them in the module's import table. Assuming a dependency of a given module actually loads at its preferred base address and has not changed, then the loader can save time by skipping the bind phase to that dependency module.

The function list can be sorted on the data in any column in the list. Simply click on the column header button for the column you wish to sort by. An arrow (▲) is displayed in the column header for the column that the list is currently sorted by. You can also size a column to its "best fit" width by double-clicking the divider line between two columns in the column header. You can search for function name text by simply typing in the first few characters of the item you wish to find.

Back to main page

The Export Function List View

The Export Function List View displays the list of export functions for the currently selected module in the Module Dependency Tree View. Export functions are functions that a module exposes to other modules. They can be thought of as the module's interface.

WinDepends uses the exported list to check for unresolved external errors in the selected module. For more information, read the Parent Import Function List View section.

While **WinDepends** scans the export list for a module, it checks each function to see if it is really a forwarded function. A forwarded function is a function that appears to be exported from a particular module, but in fact the code for the function actually lives in another module. The operating system's loader recognizes this and loads the forwarded module if necessary to resolve any imports from the parent module.

The following are the possible images that can accompany each function in the export list:

- C export function that resides in the selected module.
- C++ export function that resides in the selected module. C++ functions can be viewed in their native decorated form or in a human readable undecorated form. See Undecorate C++ Functions Command for more information.
- Ordinal export function that resides in the selected module.
- C export function that is called at least once by any module in the current module session (similar images also exist for C++ and ordinal functions).
- C export function that is called by the selected module in the Module Dependency Tree View (similar images also exist for C++ and ordinal functions). There will be a one-to-one relationship between these functions and the resolved imports in the Parent Import Function List View. You can use the Highlight Matching Item command to quickly jump between the matching import and export.
- Forwarded C export function that resides in a different module (similar images also exist for C++ and ordinal functions). The module that the function truly resides in is listed in the **Entry Point** column.

Back to main page

The Export Function View is comprised of five columns:

Image	See the above list for descriptions. The header for this column has the letter "E" in it, which just stands for "Exports"
Ordinal	The ordinal value of the exported function, if the function is exported by ordinal. This value can be "N/A" if the function is exported only by name.
Hint	The hint value for the exported function. The hint value is used internally by the operating system's loader to quickly match imports with exports. It is used as an index into the array of exported functions in the selected module.
Function	The name of the exported function, if the function is exported by name. This can be "N/A" if the function is exported only by ordinal. C++ functions can be viewed in their native decorated form or in a human readable undecorated form. See the Undecorate C++ Functions Command for more information.
Entry Point	The entry point memory address for the function. This is usually a relative offset from the base address at which the module will load at by the operating system's loader. This base address is usually the base address listed in the Module List View for the particular module. If the function is forwarded to another module, then a forward string will be displayed instead of an address. The forward string is in the form of ModuleName.FunctionName.

The function list can be sorted on the data in any column in the list. Simply click on the column header button for the column you wish to sort by. An arrow (▲) is displayed in the column header for the column that the list is currently sorted by. You can also size a column to its "best fit" width by double-clicking the divider line between two columns in the column header. You can search for function name text by simply typing in the first few characters of the item you wish to find.

Back to main page

The Log View

This view is used to log module warnings, module errors and all other program activity.

You may copy text from the Log View using Copy Command. You can also search the Log View for text using Find Command and Find Next Command.

Back to main page

The File Menu

The File menu offers the following commands:

Open...	Opens and processes a module file.
	Keys: **CTRL+O**
	Toolbar: 📂
Close	Closes the Module Session View (clears content of all lists related to currently processed file)
Save	Saves the Module Session View as **WinDepends Session** file using the same name and type that you have previously saved the file with. If you have not previously saved the Module Session View, then this command behaves just like the Save As Command, which will display the standard Windows file save dialog prompting you for a file name and file type. Within the file save dialog, you can choose to save the file as a **WinDepends Session** view (WDS) file or JSON serialized file (TXT).
	Keys: **CTRL+S**
	Toolbar: 💾
Save As...	Saves the Module Session View with a new name or type. The Save As Command always displays the file save dialog, even if you have previously saved the Module Session View using a particular name and type. This allows you to choose a new name or file type to save to. If you wish to re-save the Module Session View using the same name and type that you have previously saved the file with, then you can just use the Save Command to avoid the file save dialog.
Open New Instance	Opens a new WinDepends instance.
	Keys: **CTRL+I**

Back to main page

File 1, 2, 3, ...	Opens and processes the specified module file. **WinDepends** stores up to 32 most recently opened modules at the bottom of the File menu for your convenience. To open one of the modules listed, select the module from the menu or type the number that corresponds with the module you want to open. The maximum number of most recently opened modules is limited to 32 with a default value 10.
Exit	Exits WinDepends.

Back to main page

The Edit Menu

The Edit menu offers the following commands:

Copy	Copies the selection in the current view to the clipboard as text.
	Keys: **CTRL+C**
	Toolbar: 📋
Select All	Selects all items in the current view.
	Keys: **CTRL+A**
Find...	Finds text in Log View.
	Keys: **CTRL+F**
Find Next	Repeats last find operation in the Log View.
	Keys: **F3**
Clear Log Window	Clears the contents of the Log View in Module Session View.

Back to main page

The View Menu

The View menu offers the following commands:

System Information...	Displays information about the system. If WinDepends loaded WinDepends Session file then system information stored in this file will be displayed instead.
	Toolbar: 🖥
Expand All	Expands all nodes in the Module Dependency Tree View making entire tree visible.
	Keys: **CTRL+E**
Collapse All	Collapses all nodes in the Module Dependency Tree View, leaving only root module visible.
	Keys: **CTRL+W**
Auto Expand	When checked, the Module Dependency Tree View will automatically expand to show modules as they are loaded.
	Keys: **F8**
	Toolbar:
Full Paths	Shows or hides full file path in the Module Dependency Tree View and the Module List View. When the Full Paths option is on, both will display the complete path to each module. When this option is off, these views will display only file names. This option also effects how the Copy Command work. When the Full Paths option is on, the Copy Command will copy the full paths of the selected files to the clipboard, otherwise it just copies the file names.
	Keys: **F9**
	Toolbar: c:\

Back to main page

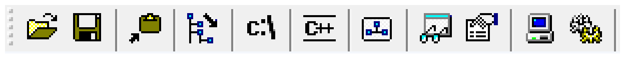
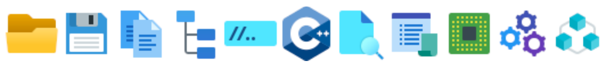
Undecorate C++ Functions	Display undecorated C++ functions names in both the Parent Import Function List View and the Export Function List View. This option requires DBGHELP.DLL to be present in system or SYMSRV.DLL and DBGHELP.DLL from Debugging Tools For Windows package. Please refer to https://docs.microsoft.com/en-us/windows/win32/debug/dbghelp-versions for information about difference between default Windows shipped dlls and these that can be obtained from **Debugging Tools For Windows** package.
	Keys: **F10**
	Toolbar:
Show Resolved API sets	Display resolved Windows ApiSet libraries in the Module Dependency Tree View and the Module List View.
	Keys: **F11**
	Toolbar:
Highlight Matching Item	Highlights the matching item in the related view.
	Keys: **CTRL+M**
Highlight Original Instance In Tree	Highlights the original instance of the selected module in the Module Dependency Tree View.
	Keys: **CTRL+K**
Highlight Previous Instance In Tree	Highlights the previous instance of the selected module in the Module Dependency Tree View.
	Keys: **CTRL+B**
Highlight Next Instance In Tree	Highlights the next instance of the selected module in the Module Dependency Tree View.
	Keys: **CTRL+N**
Refresh	Updates all views of the Module Session Window.
	Keys: **F5**
Open Module Location	Opens the selected modules location in Windows Explorer.
	Keys: **CTRL+J**

Back to main page

View Module in External Viewer	Opens the selected modules in the external module viewer. If the active view is the Module Dependency Tree View, Parent Import Function List View, or Export Function List View, then this command will launch the external viewer application with the module that is currently selected in the Module Dependency Tree View. If the Module List View has the focus, then WinDepends will launch a separate instance of the external viewer application for every module that is selected in the list.
	Keys: **Enter**
	Toolbar:
Lookup Function in External Help	Lookups the selected function in the external help collection.
Properties...	Displays the Windows Properties dialog for the selected modules. If the active view is the Module Dependency Tree View, Parent Import Function List View, or Export Function List View, then the Properties dialog will be displayed for the module that is currently selected in the Module Dependency Tree View. If the Module List View has the focus, then WinDepends will display a separate Properties dialog for every module that is selected in the list.
	Keys: **ALT+Enter**
	Toolbar:
Toolbar	Shows or hides the toolbar.
Status Bar	Shows or hides the status bar.

The Options Menu

The Options menu offers the following commands:

Configuration	Opens WinDepends configuration dialog.
	Keys: **F2**
Toolbar Theme	Allows to change currently used icon theme for program toolbar. There are two options available: Classic Dependency Walker style  Modern WinDepends style (which is better for high DPI screen resolutions) 
Configure Module Search Order...	This command will open configuration dialog at page which allows you to control how WinDepends searches your system for dependent files.
Configure Module Search Order (drivers)...	Similar to above, but does this for driver files (files with native subsystem and specific import elements such as ntoskrnl.exe, hal.dll are considered by WinDepends as drivers).
Configure Module External Viewer...	This command will open configuration dialog at page which allows you to configure the external viewer application and arguments.
Configure Handled File Extensions...	This command will open configuration dialog at page which allows you to configure the handled file extensions.
Configure ApiSets...	This command will open configuration dialog at page which allows you to configure how WinDepends will deal with API sets.
Configure Symbols...	This command will open configuration dialog at page which allows you to configure how WinDepends will use Microsoft Symbols for resolving functions that doesn't have public names.

Back to main page

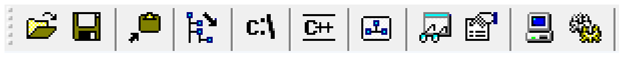
The Help Menu

The Help menu offers the following commands:

Documentation	This command will open your browser and navigate to https://github.com/hfiref0x/WinDepends.Docs where you can find the latest version of this document.
About...	This dialog displays program information, the version, and the copyright of your copy of WinDepends.

Back to main page

The Toolbar



The toolbar is displayed by default across the top of the application window, below the menu bar. The toolbar provides quick mouse access to many tools used in WinDepends.

To hide of display the Toolbar, choose the Toolbar option from the View menu.

- Opens and processes a module file. See the Open… command for more information.
- Saves the current Module Session View to a file. See the Save Command for more information.
- Copies the current selection to the clipboard as text. See the Copy Command for more information.
- When checked, the Module Dependency Tree View will automatically expand to show modules as they are added. See the Auto Expand option for more information.
- Shows or hides full path strings in the Module Dependency Tree View and the Module List View. See the Full Paths option for more information.
- Enables or disables undecoration of C++ function names in the Parent Import Function List View and the Export Function List View. See the Undecorate C++ Functions option for more information.
- Enables or disables Windows API sets redirection target resolution. See the Show Resolve API sets command for more information.
- Launches the external module viewer for the selected modules. See the View Module in External Viewer command for more information.
- Displays the Windows Properties dialog for the selected modules. See the Properties command for more information.
- Displays information about the system. See the System Information command for more information.
- Displays program configuration, see the Configuration Dialog for more information.

The toolbar may have classic (old Dependency Walker) or modern look. The meaning of commands is the same for both styles the difference is only in displayed graphics.

Back to main page

The Configuration Dialog

The Configuration Dialog is consist of the following pages:

1. Main

Use ESC key to close window	When toggled on all program dialogs will trait ESC key from keyboard as signal for exit. If ESC key is pressed over main window this will lead to program exit as well.
Auto Expand Module Tree	See Auto Expand command for more information.
Show Module Full Paths	See Full Paths for more information.
Undecorate C++ Functions	See Undecorate C++ Function command for more information.
Show Resolved API sets	See Show Resolved API sets for more information.
Upper Case Module File Names	When toggled will upper case module file names in the Module Dependency Tree View and Modules List View.
Clear Log Window on File Open	Clears the Log View window when new file is opened.
Modules Tree Depth	Sets maximum depth of Module Dependency Tree View nodes. **Warning:** increasing the maximum depth will decrease overall program performance. It is recommended to keep it in a range of 2-5 levels.
Compress Session Files	When toggled on WinDepends will use Brotli compression when saving WinDepends Session view files. Its toggled on by default and it is recommended to leave this setting as-is because this option intended mostly for debug purposes.

Back to main page

1.1. History

Most Recently Used (MRU) list depth	Can be used to configure list depth with default value of 10 and maximum value of 32.
Display full path in Most Recently Used list	When toggled on the MRU list items will display full path, when toggled off only file name will be displayed.

2. Analysis (global)

Process relocations while parsing images	When toggled on WinDepends will parse file relocations table if it present. This can however cause conflict with some specially crafted Portable Executable (PE) files. This setting is on by default.
Custom image base	Used to specify custom image base used to load PE files. Maybe useful when dealing with some specially crafted PE files. When toggled on will automatically toggle on **Process relocations while parsing images** as it depends on it.
Enable transport statistics display	Used for debug. When WinDepends will process images it will output in the Log View statistics that is come from WinDepends server such as number of send commands, total size of transferred data and overall command time.
Propagate analysis settings on dependencies	When toggled on WinDepends will apply the above global settings on all input file dependencies during processing it.
Make analysis settings default and do not ask everytime	When toggled on sets the current setting default and do not spawns file analysis configuration dialog each time when new file is opened through Open… command or drag & drop operation.

Back to main page

3. ApiSets

Retrieve from ApiSetSchema.dll file	Provides the ability to override the API set namespace used by specifying the exact DLL from which to load the namespace.
Highlight ApiSet contracts	When toggled on API set contracts will be highlighted with blue color in Module Dependency Tree View.

4. External Function Help

Search Online	Used to specify the search request used to lookup function name. Default is https://learn.microsoft.com/en-us/search/?terms=%1 where %1 represents function name.

5. External Module Viewer

Command	Command to be executed. Must specify executable file.
Arguments	Arguments that are passed to the executable file specified by “Command”.

6. Handled File Extensions

Select All	Button used to selects all predefined file types.
Associate	Button used to associate provided custom file type with WinDepends.
Elevate	Run new instance of WinDepends elevated while current instance will be closed. User must have enough privileges to be able to elevate program instance.

Back to main page

7. Module Search Order

Delete	Removes item from search order list.
Add	Adds new item to the search order list. Spawns directory selection dialog.
Collapse All	Collapses search order view nodes.
Move Up	Moves the selected search order node up, thus changing the order in which this item will be processed.
Move Down	Moves the selected search order node down, thus changing the order in which this item will be processed.

8. Server

Server Application Location	Allows specifying the server (WinDepends.Core) executable file. The program searches its own directory for this file by default.
Reconnect	Provides an ability to re-establish connection to WinDepends server. Current connection (if present) will be closed and new connection will be established. If there is no WinDepends server application running WinDepends will also try to launch server executable according to Server Application Location setting.

9. Symbols

Use debug symbols (PDB) to resolve anonymous function names when displaying module import/export list	When toggled on WinDepends will attempt to resolve DBGHELP functions required for Microsoft Symbols resolving and if succeeded use them while displaying import/export list if needed.
Dbghelp.dll path	Path to the DBGHELP.DLL file.

Back to main page

Symbols path	Path to directory where symbols will be stored during work.
Resolved Symbol Highlight Color	You can specify color used when highlighting items resolved with help of symbols.
Set Defaults	When this button is pressed default settings will be applied to dbghelp path location (defaulted to %SystemRoot%\system32), Symbols path (defaulted to %Temp%), highlight color (defaulted to yellow).

Back to main page