Dependency Walker Help Contents

- [What's New in Version 2.1](#)
- [Frequently Asked Questions (FAQ)](#)

Overview of Dependency Walker

- [Why Use Dependency Walker?](#)
- [Using Dependency Walker for Trouble Shooting Modules](#)
- [Using Dependency Walker for General Information about Modules](#)
- [Command Line Options and Return Values](#)
- [Overview of Module Version Numbers](#)
- [Types of Dependencies Handled By Dependency Walker](#)
- [Using Application Profiling to Detect Dynamic Dependencies](#)
- [How to Interpret Warnings and Errors in Dependency Walker](#)
- [Dependency Walker Path (DWP) Files](#)

Understanding the Module Session

- [The Module Session Window](#)
- [The Module Dependency Tree View](#)
- [The Module List View](#)
- [The Parent Import Function List View](#)
- [The Export Function List View](#)
- [The Log View](#)

Menus and Toolbar

- [The File Menu](#)
- [The Edit Menu](#)
- [The View Menu](#)
- [The Options Menu](#)
- [The Profile Menu](#)
- [The Window Menu](#)
- [The Help Menu](#)
- [The Toolbar](#)

What's New in Version 2.1

◘   Support for Side-by-Side versioning of modules.   This is a new feature introduced with Windows XP that allows applications to specify specific versions and/or locations of files it wishes to use.

◘   Integration with Visual Studio help, MSDN help, and MSDN online to provide the ability to display detailed help for any known function.

What was New in Version 2.0

◘   Detection of dynamically loaded modules, including details about which module actually called LoadLibrary to dynamically load the module.

◘   Detection of dynamically called functions, including details about which module actually called GetProcAddress to obtain the function address.

◘   Detection of delay-load dependencies. This is a new type of dependency that was introduced with Microsoft Visual C++ 6.0. They work on Windows 95/98/Me and Windows NT/2000/XP.

◘   Support for 64-bit Windows modules.

◘   Console mode that allows Dependency Walker to be ran without its graphical interface being displayed.   This is useful for batch files and unattended automation of Dependency Walker features.

◘   Command line options to configure module search order, column sorting, output files, profiling, and other settings.

◘   Ability to monitor module entrypoints (like DllMain) looking for module initialization failures.

◘   C++ function name undecorating to provide human readable C++ function prototypes including function names, return types, and parameter types.

◘   User definable module search paths with support for "KnownDLLs" and the "App Paths" registry keys.   Search paths can be saved and loaded from within the graphical interface or from the command line.

◘   Ability to save a module's session to a text report file for easy viewing in any text viewer.

◘   Ability to save a module's session to a comma separated value (CSV) file for easy importing into other applications.

◘   Ability to save a snapshot of an entire module session to an image file, which can be loaded by Dependency Walker at a later time on any computer.

◘   Module profiling to detect dynamic dependencies, child processes, thread activity, and exceptions. Child processes can also be profiled for their dependencies.

◘   Ability to control what file extensions Dependency Walker will add the "View Dependencies" menu item to a file's context menu in explorer.

◘   Added hotkeys to help match imports with exports, and modules in the list view with modules in the tree view.   Also added hotkeys to locate the previous, next, or original instance of a module in the tree view.

◘	Added some new columns to the Module List View. They include Link Time Stamp, Link Checksum, Real Checksum, Symbols, Actual Base, Virtual Size, and Load Order.

◘	Added an OS Information dialog. This information is also saved to text and Dependency Walker Image (DWI) files.

◘	All list views can now be sorted by icon, which provides an easy way of grouping items of similar type.

◘	You can now search all list views for text by simply typing in a few characters to match in the currently sorted column.

◘	Added color-coding to the module list view and log view to help highlight problems.

Why Use Dependency Walker?

Have you ever...

- ...wondered why an application or module was failing to load?
- ...wondered what minimum set of files are required to run a particular application or load a particular DLL?
- ...wondered why a certain module was being loaded with a particular application?
- ...wanted to know what functions are exposed by a particular module, and which ones are actually being called by other modules?
- ...wanted to know the parameter and return types of exported C++ functions?
- ...wanted to remove all dependencies for a given module?
- ...wanted to know the complete path of all the modules being loaded for a particular application?
- ...wanted to know all the base addresses of each module being loaded for a particular application?   What about versions?   Or maybe CPU types?
- ...received one of the following errors...
- The dynamic link library BAR.DLL could not be found in the specified path...
- The procedure entry point FOO could not be located in the dynamic link library BAR.DLL.
- The application or DLL BAR.DLL is not a valid Windows image.
- The application failed to initialize properly.
- Initialization of the dynamic link library BAR.DLL failed. The process is terminating abnormally.
- The image file BAR.EXE is valid, but is for a machine type other than the current machine.
- Program too big to fit in memory.

Using Dependency Walker for Trouble Shooting Modules

Dependency Walker recursively scans all dependent modules required by a particular application.   During this scan it performs the following tasks:

■   Detects missing files.   These are files that are required as a dependency to another module.   A symptom of this problem is the "The dynamic link library BAR.DLL could not be found in the specified path..." error.

■   Detects invalid Files.   This includes files that are not Win32 or Win64 compliant and files that are corrupt.   A symptom of this problem is the "The application or DLL BAR.EXE is not a valid Windows image" error.

■   Detects import/export mismatches.   Verifies that all functions imported by a module are actually exported from the dependent modules.   All unresolved import functions are flagged with an error.   A symptom of this problem is the "The procedure entry point FOO could not be located in the dynamic link library BAR.DLL" error.

■   Detects circular dependency errors.   This is a very rare error, but can occur with forwarded functions.

■   Detects mismatched CPU types of modules.   This occurs if a module built for one CPU tries to load a module built for a different CPU.

■   Detects checksum inconsistencies by verifying module checksums to see if any modules have been modified after they were built.

■   Detects module collisions by highlighting any modules that fail to load at their preferred base address.

■   Detects module initialization failures by tracking calls to module entrypoints and looking for errors.

Dependency Walker can also perform a run-time profile of your application to detect dynamically loaded modules and module initialization failures.   The same error checking from above applies to dynamically loaded modules as well.

Using Dependency Walker for General Information about Modules

Dependency Walker is more than just a trouble shooting utility.   It also provides a great deal of valuable information about the module layout of a particular application and details on each module.   Dependency Walker provides the following information:

- 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.
- 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.
- 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 actual base load address of the module.
- The virtual size of the module.
- The load order of the module with respect to other modules.
- 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.

Command Line Options and Return Values

DEPENDS.EXE   [/?] [/c] [/a:#] [/f:#] [/u:#] [/ps:#] [/pp:#] [/po:#] [/ph:#] [/pl:#] [/pg:#] [/pt:#] [/pn:#] [/pe:#] [/pm:#] [/pf:#] [/pi:#] [/pc:#] [/pa:#] [/pd:dir] [/pb] [/sm:#] [/si:#] [/se:#] [/sf:#] [/od:path] [/ot:path] [/of:path] [/oc:path] [/d:path] [path [args...]]

/?   **Help** - Displays this page.

/c   <u>C</u>**onsole mode** - Dependency Walker will process the other command line options and exit without displaying its graphical interface.   You must specify a module or Dependency Walker Image (DWI) file to open when using this option.

/a:#   **Auto Expand** - Use /a:0 to start Dependency Walker with the <u>Auto Expand</u> setting initially turned off, or /a:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/f:#   **View full paths** - Use /f:0 to start Dependency Walker with the <u>View Full Paths</u> setting initially turned off, or /f:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/u:#   **Undecorate C++ functions** - Use /u:0 to start Dependency Walker with the <u>Undecorate C++ Functions</u> setting initially turned off, or /u:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/ps:#   **Profiling option: Simulate ShellExecute by inserting any App Paths directories into the PATH environment variable** - Use /ps:0 to start Dependency Walker with this setting initially turned off, or /ps:1 to start with it turned on. If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pp:#   **Profiling option: Log DllMain calls for process attach and process detach messages** - Use /pp:0 to start Dependency Walker with this setting initially turned off, or /pp:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/po:#   **Profiling option: Log DllMain calls for all other messages, including thread attach and thread detach** - Use /po:0 to start Dependency Walker with this setting initially

turned off, or /po:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/ph:#   **<u>P</u>rofiling option: Hook the process to gather more detailed dependency information** - Use /ph:0 to start Dependency Walker with this setting initially turned off, or /ph:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pl:#   **<u>P</u>rofiling option: Log LoadLibrary function calls** - Use /pl:0 to start Dependency Walker with this setting initially turned off, or /pl:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.   If this option is turned on, then the "Hook the process to gather more detailed dependency information" option will also be turned on.

/pg:#   **<u>P</u>rofiling option: Log GetProcAddress function calls** - Use /pg:0 to start Dependency Walker with this setting initially turned off, or /pg:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.   If this option is turned on, then the "Hook the process to gather more detailed dependency information" option will also be turned on.

/pt:#   **<u>P</u>rofiling option: Log thread information** - Use /pt:0 to start Dependency Walker with this setting initially turned off, or /pt:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pn:#   **<u>P</u>rofiling option: Use simple thread numbers instead of actual thread IDs** - Use /pn:0 to start Dependency Walker with this setting initially turned off, or /pn:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used. If this option is turned on, then the "Log thread information" option will also be turned on.

/pe:#   **<u>P</u>rofiling option: Log first chance exceptions** - Use /pe:0 to start Dependency Walker with this setting initially turned off, or /pe:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pm:#   **<u>P</u>rofiling option: Log debug output messages** - Use /pm:0 to start Dependency Walker with this setting initially turned off, or /pm:1 to start with it turned on.   If this option is

not specified, then the setting from the last time you ran Dependency Walker will be used.

/pf:#   **<u>P</u>rofiling option: Use full paths when logging file names** - Use /pf:0 to start Dependency Walker with this setting initially turned off, or /pf:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pi:#   **<u>P</u>rofiling option: Log a time stamp with each line of log** - Use /pi:0 to start Dependency Walker with this setting initially turned off, or /pi:1 to start with it turned on.   If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.

/pc:#   **<u>P</u>rofiling option: Automatically open and profile child processes** - Use /pc:0 to start Dependency Walker with this setting initially turned off, or /pc:1 to start with it turned on. If this option is not specified, then the setting from the last time you ran Dependency Walker will be used.   This option is ignored when running in console mode.

/pa:#   **<u>P</u>rofiling option: Turn all profiling options on or off** - Use /pa:0 to initially turn all profiling options off, or /pa:1 to initially turn them all on.   This option can be used before other profiling options.   For example, /pa:1 /pf:0 will turn on all options except for the "Use full paths when logging file names" option.

/pd:dir   **<u>P</u>rofiling option: Starting directory** - Specifies the starting directory to use when profiling the module.   This option requires that you specify a module to open.

/pb   **<u>P</u>rofiling option: Automatically begin profiling after the module has been loaded** - This option requires that you specify a module to open.   If an output option (/od, /ot, /of, or /oc) is specified, Dependency Walker will wait until the profiling fully completes before saving the results.

/sm:#   **<u>S</u>ort column for module list view** - This option controls the initial sort column that Dependency Walker will use when sorting the items in the <u>Module List View</u>.   If this option is not specified, then the value from the last time you ran Dependency Walker will be used.   The values allowed are:

1. Icon
2. Module Name or Path
3. File Time Stamp
4. Link Time Stamp
5. File Size
6. File Attributes

7. Link Checksum
8. Real Checksum
9. CPU Type
10. Subsystem Type
11. Symbol Types
12. Preferred Base Address
13. Actual Base Address
14. Virtual Size
15. Load Order
16. File Version
17. Product Version
18. Image Version
19. Linker Version
20. OS Version
21. Subsystem Version

/si:#   **<u>S</u>ort column for parent import function list view** - This
option controls the initial sort column that Dependency Walker
will use when sorting the items in the <u>Parent Import Function
List View</u>.   If neither this option or the /sf option is specified,
then the value from the last time you ran Dependency Walker
will be used.   The values allowed are:

1. Icon
2. Ordinal Value
3. Hint Value
4. Function Name
5. Entry Point Address

/se:#   **<u>S</u>ort column for export function list views** - This option
controls the initial sort column that Dependency Walker will
use when sorting the items in the <u>Export Function List View</u>.
If neither this option or the /sf option is specified, then the
value from the last time you ran Dependency Walker will be
used.   The values allowed are:

1. Icon
2. Ordinal Value
3. Hint Value
4. Function Name
5. Entry Point Address

/sf:#   **<u>S</u>ort column for both function list views** - This option
controls the initial sort column that Dependency Walker will
use when sorting the items in both the <u>Parent Import Function
List View</u> and the <u>Export Function List View</u>.   If no sort column
option is specified for a particular column, then the value(s)
from the last time you ran Dependency Walker will be used.
The values allowed are:

1. Icon
2. Ordinal Value
3. Hint Value
4. Function Name
5. Entry Point Address

/od:path   **Output file in Dependency Walker Image (DWI) format** - This option requires that you specify a module or Dependency Walker Image (DWI) file to open.   Once the module has been processed, the results will be written to the specified file in the Dependency Walker Image (DWI) format.

/ot:path   **Output file in text format** - This option requires that you specify a module or Dependency Walker Image (DWI) file to open.   Once the module has been processed, the results will be written to the specified file in text format.

/of:path   **Output file in text format with import / export function lists** - This option requires that you specify a module or Dependency Walker Image (DWI) file to open.   Once the module has been processed, the results will be written to the specified file in text format, including the import and export function lists.

/oc:path   **Output file in Comma Separated Value (CSV) format** - This option requires that you specify a module or Dependency Walker Image (DWI) file to open.   Once the module has been processed, the results will be written to the specified file in a Comma Separated Value (CSV) format.

/d:path   **Dependency Walker Path (DWP) file to load** - This options allows you to specify a Dependency Walker Path (DWP) File to load and use as the initial search path when searching for modules.   DWP files can be created using the Configure Search Order command in Dependency Walker.

path   **Path to a module or Dependency Walker Image (DWI) file to load** - For this option, you can specify a file name, a relative path, or a full path to a file to load.   The file must be a 32-bit or 64-bit Windows module or a Dependency Walker Image (DWI) file.   This path must come after any options intended for Dependency Walker since all options that follow this path are assumed to be program arguments for use when profiling the module.

args...   **Program arguments** - Specifies the command line arguments to use when profiling the module specified by the **path** option.   Dependency Walker considers any text following the **path** option as being program arguments.   For this reason, any options intended for Dependency Walker

must be specified before the **path** option.   If the file specified by the **path** option is really a Dependency Walker Image (DWI) file, then the args are ignored.

General Rules about Command Line Options

- Options are case insensitive.   For example, "/c" and "/C" are equivalent.
- Options may start with a slash or a dash.   For example, "/c" and "-c" are equivalent.
- The colons (:) shown in the options above are optional.   They may be removed or replaced with spaces.   For example, "/f:0", "/f 0", and "/f0" are equivalent.
- All profiling options are cumulative from left to right.   For example, /pa:1 /pm:0 will turn on all the profiling options, then turn off the "Log debug output messages" option, but /pm:0 /pa:1 will simply turn on all profiling options.
- Program options intended for Dependency Walker must come before the module path.   All options after the module path will be passed to the module as its command line when profiled.
- If you wish to specify text that has spaces, that text should be placed in quotes.   For example:

      depends /pb /oc "c:\output files\foo bar.csv" "c:\input files\foo bar.exe" 1 2 3 "this is a test"

- Multiple options can be grouped together.   You may even append options to other options that require numerical values.   The only options that cannot be appended to are options that require a path or text values (/pd, /od, /ot, /of, /oc, and /d).   For example:

      depends /c /f:0 /u:1 /pa:1 /pf:0 /pe:0 /pb /sm:12 /sf:4 /d:search.dwp /oc:result.csv /od:result.dwi foo.exe

   Could be shortened to:

      depends /cf0u1pa1pf0pe0pbsm12sf4dsearch.dwp /ocresult.csv /odresult.dwi foo.exe bar

- All options can be specified with or without the "Console Mode" option (/c).
- More than one output file type option can be specified.

Return Values

When Dependency Walker exits, it returns a set of bit flags that are OR'ed together.   There are three groups of error flags - module warnings, module errors, and processing errors.   The error flags have been

arranged in a way that makes it easy to detect the severity of a problem.

If the return value is greater than or equal to 0x00010000, then there was a processing error with Dependency Walker and no work was done. Otherwise, if the return value is greater than or equal to 0x00000100, then the operating system will not be able to load the module due to some module or dependency error.   Otherwise, if the return value is greater than or equal to 0x00000001, then the module has no load-time dependency problems and will most likely have no problems loading, but may have runtime problems.

Module Warnings - Application should load, but might fail during runtime.

0x00000001	At least one dynamic dependency module was not found.
0x00000002	At least one delay-load dependency module was not found.
0x00000004	At least one module could not dynamically locate a function in another module using the GetProcAddress function call.
0x00000008	At least one module has an unresolved import due to a missing export function in a delay-load dependent module.
0x00000010	At least one module was corrupted or unrecognizable to Dependency Walker, but still appeared to be a Windows module.
0x00000020	At least one module failed to load during profiling. This usually occurs when a module returns 0 from its DllMain function or generates an unhandled exception while processing the DLL_PROCESS_ATTACH message.

Module Errors - Application will fail to load by the operating system.

0x00000100	At least one file was not a 32-bit or 64-bit Windows module.
0x00000200	At least one required implicit or forwarded dependency was not found.
0x00000400	At least one module has an unresolved import due to a missing export function in a dependent module.
0x00000800	Modules with different CPU types were found.
0x00001000	A circular dependency was detected.
0x00002000	There was an error in a Side-by-Side configuration file.

Processing Errors - All or some modules could not be processed.

0x00010000	There was an error with at least one command line option.
0x00020000	The file you specified to load could not be found.
0x00040000	At least one file could not be opened for reading.
0x00080000	The format of the Dependency Walker Image (DWI) file was unrecognized.
0x00100000	There was an error while trying to profile the application.
0x00200000	There was an error writing the results to an output file.
0x00400000	Dependency Walker ran out of memory.
0x00800000	Dependency Walker encountered an internal program error.

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.

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 3.0" is a set of ten utilities, including "Acme Virus Checker version 1.5".   The virus checker executable might have a file version of 1.5.0.0 and a product version of 3.0.0.0

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.

Dependency Walker 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 Dependency Walker and not the Windows Properties dialog.

A great web site for looking up version numbers of modules is the Microsoft DLL Help Database (http://support.microsoft.com/servicedesks/FileVersion/dllinfo.asp).   This site has detailed version histories of DLLs and lists what products were shipped with each version.   This database can be helpful in tracking down version

problems.

Types of Dependencies Handled By Dependency Walker

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.

Dependency Walker fully supports modules loaded by all of the above techniques.   Case 1, 2, and 3 can easily be detected by just opening a module in Dependency Walker.   Case 4 and 5 require runtime profiling, a new feature in Dependency Walker 2.0.   For more information on profiling, see the <u>Using Application Profiling to Detect Dynamic Dependencies</u> section.

Using Application Profiling to Detect Dynamic Dependencies

Dependency Walker version 2.0 adds application profiling, a technique used to watch a running application to see what modules it loads.   This allows Dependency Walker to detect dynamically loaded modules that are not necessarily reported in any on the import tables of other modules.   Dependency Walker's profiler can also detect when a module fails to initialize, which often results in the "The application failed to initialize properly" error.

When a module is first opened by Dependency Walker, it is immediately scanned for all implicit, delay-load, and forwarded dependencies (for more information on dependency types, see the <u>Types of Dependencies Handled By Dependency Walker</u> section).   Once all the modules have been scanned, the results are displayed.   In addition to these known dependencies, modules are free to load other modules at run-time without any prior warning to the operating system.   These types of dependencies are known as dynamic or explicit dependencies.   There is really no way to detect dynamic dependencies without actually running the application and watching it to see what modules it loads at run-time.   This is exactly what Dependency Walker's application profiling does.

For profiling to work, the module you open in Dependency Walker has to be an executable file (usually ends with .EXE) that is designed to run on the system you are working with.   If not, the <u>Start Profiling</u> menu option and toolbar button will not be enabled.   When you choose to profile an application, your application should begin to run.   As your application runs, Dependency Walker will gather information and log it to the <u>Log View</u>, as well as update the other views.

It is the job of the user to "exercise" the application to ensure that all dynamic dependencies are found.   Usually dynamic dependencies are only loaded when needed.   For example, modules related to printing might only be loaded if the application actually prints.   In a case like this, if the application does not perform a print while being profiled, then Dependency Walker will not detect those modules related to printing.   Other modules might only get loaded if an error occurs in the application.   Scenarios like these might be hard to produce.   Because of this, **It is impossible to guarantee that all dynamic dependencies are found**, but the more an application is exercised, the better the odds are of finding them.

Dependency Walker's application profiler tracks every module that gets loaded and attempts to determine which module actually requested the file to be loaded.   This allows dynamically loaded modules to be inserted into

the Module Dependency Tree View as a child of the module that actually loaded the module.

The profiler works by hooking particular function calls in the remote process being profiled.   On Windows 95, Windows 98, and Windows Me, only non-system modules can be hooked.   The result is that when a system module dynamically loads another module, the profiler cannot tell who the parent module is for the dynamically loaded module.   Parentless modules like these will be added to the root of the Module Dependency Tree View.   All modules that are loaded due to a system-wide hook will also be added to the root of the Module Dependency Tree View since these types of modules are loaded directly by the OS and have no parent module.   Even though Dependency Walker may not be able to detect the parent of a dynamic dependency, it does guarantee that all modules that get loaded by the application will be detected.

One final benefit of the profiler is that it can correct the paths of any modules that may have been incorrectly determined during the initial implicit module scan.   When you first open a module in Dependency Walker, it recursively scans all the import and export tables of modules to build the initial module hierarchy.   Only file names are stored in these tables, so dependency walker uses the rules you have set up in the Module Search Order Dialog to determine the full path to each module.   During profiling, Dependency Walker examines the real path of each module as they load and compares them to the modules in the tree.   If a module loads from a different path than Dependency Walker expected it to load from, then it will update the module hierarchy and other views to reflect the change.

How to Interpret Warnings and Errors in Dependency Walker

Dependency Walker may generate many warnings and errors for an application.   Some errors may cause an application to fail, while others are harmless and can be ignored.   Most failures fit into one of two categories: load-time failures or run-time failures.

A load-time failure means that an application or module didn't even have a chance to run.   In more technical terms, this usually means that the entry-point to a module was never called since the operating system couldn't load all the required modules.   This can occur if an implicit or forward dependency could not be found or was missing a needed function (for more information on dependency types, see the <u>Types of Dependencies Handled By Dependency Walker</u> section).   You will also encounter a load-time failure if the application attempts to load a corrupt or non-Windows module, a module for a different CPU type then you are using, or a 16-bit module into a 32-bit application.   Here are some common load-time error messages:

- The dynamic link library BAR.DLL could not be found in the specified path…
- The procedure entry point FOO could not be located in the dynamic link library BAR.DLL.
- The application or DLL BAR.DLL is not a valid Windows image.
- The application failed to initialize properly.
- Initialization of the dynamic link library BAR.DLL failed. The process is terminating abnormally.
- The image file BAR.EXE is valid, but is for a machine type other than the current machine.

Most load-time problems can be immediately detected by Dependency Walker.   When you first open a module in Dependency Walker, it scans that module for all implicit, forward, and delay-load dependencies.   Implicit and forward dependencies are required by the operating system in order for the application to run.   If any implicit or forward dependencies are missing or have errors, then it is likely that the application will encounter a load-time failure if run.   Delay-load dependencies are not required by the operating system at load-time, so errors or warning with delay load dependencies may or may not cause problems.

Run-time dependencies are modules that an application loads after it has initialized and begun to run.   This is usually achieved by calling one of the LoadLibrary type functions.   Once a module has been loaded, an application can call the GetProcAddress function to locate a specific function in the newly loaded module.   Dependency Walker can track all these calls and reports

any failures.   However, if the application is prepared to handle the failure, then the warning can be ignored.

There are many reasons for using run-time dependencies.   First, they can increase load-time performance since an application can delay the loading of certain modules that may not be needed until later.   For example, if an application uses a DLL related to printing, that DLL might not get loaded unless you actually print something from the application.   Second, they can be used in cases where a module, or a function within a module, may not exist.   For example, an application might need to call a Windows NT specific function when running on Windows NT, but the module or function does not exist on Windows 9x.   If the application were to implicitly link to the module that the function lives in, then a load-time failure would occur on Windows 9x since the operating system would not be able to locate the function at load-time.   By making it a run-time dependency, the application can check to see if the function exists and only call it if it does.

There are two types of run-time dependencies: explicit dependencies (often referred to as dynamic dependencies) and delay-load dependencies. Explicit dependencies can be loaded at anytime during the life of the application with no prior notice.   Because of this, the only way to determine what explicit dependencies an application will use is to run the application and watch it to see what it loads (for more information on profiling, see the Using Application Profiling to Detect Dynamic Dependencies section).   With explicit dependencies, the application directly calls LoadLibrary and GetProcAddress to do the work.

Delay-load dependencies are actually implemented as explicit dependencies, but a helper library and the linker do most of the work.   Most all Windows modules have an "import table" stored in them.   This table is built by the linker and used by the operating system to determine the implicit and forward dependencies of a given module.   Any module or function in this list that cannot be found will cause the module to fail.   If you tell the linker to make a module a delay-load dependency, then instead of storing that module's information in the main import table, it stores it in a separate delay-load import table.   At run-time, if a module calls into a delay-load dependency module, the call is trapped by the helper library.   This library then uses LoadLibrary to load the module and GetProcAddress to query all the functions referenced in the module.   Once this is complete, the call is passed along to the real function and execution resumes without the module that made the call even knowing what just happen.   All future calls from that specific module to the delay-loaded module will be made directly into the already loaded module instead of being trapped by the helper library.

The delay-load helper library has a mechanism for notifying the caller if there is a failure.   Like failures with explicit dependencies, if the application is

prepared for the failure, then this should not be a problem.

To summarize, implicit and forward dependencies are required dependencies that need to exist and have no errors or warnings.   Explicit and delay-load dependencies may not need to exist and may not need to export all the functions that the parent module wishes to import from them.   However, if an application is not prepared to handle a missing explicit or delay-load module, or a missing function within an explicit or delay-load module, then this can result in a run-time failure of the application.   Dependency Walker cannot predict if an application plans to handle failures, so it just warns you of all potential problems.   If you find an application runs smoothly, then you can probably ignore most all warnings.   However, if your application were to fail, then the warnings may provide some insight as to what caused the failure.

There is one other type of warning generated by Dependency Walker while profiling that is worth mentioning.   This is related to first and second exceptions.   When an exception (like an access violation) occurs in an application, the application is given a chance to handle the exception. These are known as first chance exceptions.   If the application handles the exception, then there should be no problem and the exception can probably be ignored.   If the application does not handle the first chance exception, then it turns into a second chance exception, which are usually fatal to the application.   When a second chance exception occurs, the operating system usually puts up a dialog telling you that the application has crashed and needs to exit.

Dependency Walker always logs second chance exceptions and can optionally log first chance exceptions.   Many applications routinely generate first chance exceptions and handle them.   This is not a sign of a bad application since there are many legitimate reasons to generate first chance exceptions and handle them.

Dependency Walker Path (DWP) Files

Dependency Walker Path (DWP) files are used to define how Dependency Walker locates modules on your system.   By default, Dependency Walker is set up to simulate the search algorithm that the operating system uses to locate modules.   However, you can override this default and set up your own custom search criteria.   See the Module Search Order Dialog section for more information.

DWP files are usually created by configuring a search order in the Module Search Order Dialog, and then choosing save from that dialog to save the search order to a DWP file.   This DWP file can then be loaded at a later time from the Module Search Order Dialog or from the Command Line.

DWP files can also be created and edited by hand.   DWP files are simply text files that contain a list of search groups.   The following is a list of supported keywords:

SxS	Side-by-Side components
KnownDLLs	The system's "KnownDLLs" list
AppDir	The application directory
32BitSysDir	The 32-bit system directory
16BitSysDir	The 16-bit system directory (Windows NT/2000/XP only)
OSDir	The system's root OS directory
AppPath	The application's registered "App Paths" directories
SysPath	The system's "PATH" environment variable directories
UserDir	A user defined directory

Each keyword must be on a line by itself.   All keywords are case insensitive. Except for the UserDir keyword, no keyword can be specified more than once.   The UserDir keyword is a special keyword that also requires a directory path.   The syntax for it is:

    UserDir c:\path\to\some\directory\

You may use system variables in the path as well.   For example:

    UserDir %build_directory%\%target_cpu%\debug\

All spaces and empty lines in the DWP file are ignored, except for spaces that are part of a directory path.   No quotes should be used with any of the keywords or paths.   You may add comments to the file by starting a line with a colon (:), semicolon (;), forward slash (/), single quote ('), or pound (#).

Frequently Asked Questions (FAQ)

Q: Dependency Walker 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 Dependency Walker, it only shows implicit, forwarded, and delay-load dependencies.   Many dependencies are loaded dynamically and will not be detected until you profile the application.   For more information, see <u>Types of Dependencies Handled By Dependency Walker</u> and <u>Using Application Profiling to Detect Dynamic Dependencies</u>.

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)**.   Dependency Walker supports C++ undecoration by using the <u>Undecorate C++ Functions Command</u>.

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 project and declare the actual function names you want exported.

Q: My application seems to run just fine during profiling, however, I see errors in the log view and red or yellow icons in the other views.   Is this normal?

A: It is fairly normal to see errors or warnings during profiling.   One common error seen is when one module tries to dynamically load another module (using one of the LoadLibrary functions), but the module is not found.   Dependency Walker makes a note of this failure, but if the application is prepared for the failure, then this is not a problem.   Another common error is when a module tries to dynamically locate a function (using GetProcAddress) in a module.   Again, this is not a problem if the application is prepared for the failure.   You may also see first-chance exceptions occur in the log view.   If the application handles the exceptions and they don't turn into second-chance exceptions, then this is not a problem.   All these cases are normal, and can usually be ignored.   However, if the application you are profiling crashes or fails to run properly, then the errors may provide some insight as to what caused the problem.   See the How to Interpret Warnings and Errors in Dependency Walker section for more details.

Q: Wow, my application depends on all those files?   Which ones do I need to redistribute with my application?

A: For starters, there are certain modules you should never redistribute with your application, such as kernel32.dll, user32.dll, and gdi32.dll.   To see which files you are allowed to redistribute, you can look for a file named REDIST.TXT on your development computer.   This file is included with development suites like Microsoft Visual C++ and Visual Basic.   You can also look up "redistributable files" and "redist.txt" in the MSDN index for more information on what files to redistribute, how to redistribute them, how to check file versions, etc.   Another site worth mentioning is the Microsoft DLL Help Database (http://support.microsoft.com/servicedesks/FileVersion/dllinfo.asp).   This site has detailed version histories of DLLs, and lists what products were shipped with each version.

Q: What does "Shared module not hooked" mean, and why are some module's DllMain calls never being logged?

A: Dependency Walker hooks modules as they load in order to track calls to functions like DllMain, LoadLibrary, and GetProcAddress.   Any module loaded above address 0x80000000 (usually system modules) on Windows 95/98/Me is shared system-wide and cannot be hooked.   The result is that Dependency Walker cannot log information about function calls in those modules.   Windows NT/2000/XP does not have this limitation.   See Using Application Profiling to Detect Dynamic Dependencies for more information.

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

A: Dependency Walker 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, and a dynamic dependency, 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: Is there a command line version of Dependency Walker?

A: Dependency Walker can be run as a graphical application or as a console application.   When the console mode option is used, Dependency Walker can process a module, save the results, and exit without any graphical interface or user prompting.   See the Command Line Options section for more information.

Q: Will Dependency Walker work with Visual Basic or Windows CE modules?

A: Yes.   Dependency Walker will work with any 32-bit or 64-bit Windows module, regardless of what language was used to develop it or what Windows platform it is targeted for.   This includes modules for Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, Windows XP, and Windows CE.

Q: Will Dependency Walker work with 16-bit modules?

A: No.   Dependency Walker currently only supports 32-bit and 64-bit Windows modules.

Q: What do all the version numbers mean?

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

Q: Can I print out the results of a session?

A: No, but you can save the results to several different text formats which can be viewed or printed from a text viewer program like Notepad.

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

A: Dependency Walker supports several ways to capture the data in a session.   All the views support simple copying from them using the Copy Command.   Dependency Walker also supports several methods of saving the entire session to a file.   There are various text formats that can be easily printed or emailed to someone for viewing.   You can also save the results to a Dependency Walker Image (DWI) file, which can be loaded by Dependency Walker 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 and File Save Dialog section.

Q: What do all the icons mean?

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

Q: Can I search for a function by name or ordinal?

A: All the list views in Dependency Walker can be sorted and searched. Any text you type while in a list view will search for that text in the column that the list is currently sorted by.   For example, if the export function list is sorted by function names and you type "Get", the first function that starts with "Get" will be highlighted.   This will work for any column in any list.   For more details, see the help sections for the actual list views.

Q: Dependency Walker's open dialog is not showing a file that I want to open.   How can I fix this?

A: By default, Windows "hides" certain system files (like DLLs) from the user.   To change this setting, open "My Computer" and select "Options" from the menu.   Depending on what version of Windows you are using, this should be off of the "View" or "Tools" menu, and may be called "Folder Options" or just "Options".   In the dialog that appears, choose the "View" tab.   You should see an option that reads either "Show all files" or "Show hidden files and folders".   Make sure this option is selected.   You will also see a check-box that reads "Hide MS-DOS file extensions for file types that are registered" or "Hide file extensions for known file types".   You will want to uncheck this box.   Once done, press "Ok" in that dialog.   Dependency Walker should now show all system files in its open dialog.

Q: How do I uninstall Dependency Walker?

A: Dependency Walker 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 Dependency Walker to handle certain file extensions, you will probably want to remove those associations before deleting the program.   This can be done by using the Handled File Extensions command.   The files to delete when Dependency Walker is no longer needed are depends.exe, depends.dll, depends.cnt, and depends.hlp.

Q: Why are some modules looking for a function named "IsTNT" in KERNEL32.DLL?

A: TNT is a 32-bit emulation layer written by Phar Lap.   There are still some modules in use that have pieces of code that check to see if they are running on TNT by calling GetProcAddress("IsTNT") for KERNEL32.DLL.

This warning can be ignored.

Q: Why are some modules trying to load a module named "AUX"?

A: This is usually related to modules trying to load the AUX audio driver. Since AUX is a reserved DOS name, the load fails. This warning is harmless and can be ignored.

Q: MFC42.DLL is trying to load MFC42LOC.DLL, but it is not found. [or] COMCTL32.DLL is trying to load CMCTLENU.DLL, but it is not found. Why is this?

A: Both MFC42LOC.DLL and CMCTLENU.DLL are language specific resource DLLs that may not be needed on your system. Many modules on Windows store all their language specific messages in external DLLs (one per language). At run-time, the module loads the language DLL for the current language of the operating system. The names of the modules usually end in "ENU" for United States English, "ESP" for Spanish, "JPN" for Japanese, etc. The "LOC" ending that MFC uses stands for "localized". When MFC is installed, it copies the correct language DLL to your system and renames it to MFC42LOC.DLL. So, why the missing module? Well, most modules protect themselves from failure by storing one default language in the main DLL itself. It the language specific resource DLL fails to load, then the module defaults to using the local resources in itself. In most cases, these default resources are the same resources as would be in the ENU version of the resource DLL. For this reason, there does not need to be an ENU version of the resource DLL, and therefore it fails to find one at runtime. This is normal.

Module Session Window

A module session window is created for every module or Dependency Walker Image (DWI) file that is opened.   The window is split into the following five views:

- Module Dependency Tree View
- Module 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.   All views support context help.   You may press F1 anywhere in Dependency Walker to get help on the item that currently has the focus. You may also use the Context Help tool to allow you to simply click on the item you wish to get help on.

For navigating through the views, see the Previous Pane command and the Next Pane command.   For navigating through the open Module Session Windows, see the Previous Window command, the Next Window command, and the Window 1, 2, 3, ... command.

Module Dependency Tree View

The Module Dependency Tree View displays a hierarchical view of all the modules' dependencies.   There are several ways a module can be a dependency of another module.   For more information on dependency types, see the Types of Dependencies Handled By Dependency Walker section.

Dependency Walker starts with the root module you chose to open and scans its import tables to build a list of required dependent modules.   Dependency Walker then scans each of these dependent modules for their dependent modules.   This recursion continues until all modules and their dependent modules have been processed.

To prevent a bloated tree and possible infinite circular loops with dependent modules, Dependency Walker stops processing a given branch of the tree when it reaches a module that it has already processed somewhere else in the tree.   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 Dependency Walker had processed it, use the Highlight Original Instance Command to find the original instance of the module in the tree.

Dependency Walker also scans each dependent module looking for forwarded function calls to other modules.   If a forwarded function is found and actually called by the parent module, then the module that the function is forwarded to is also pulled in and added to the dependency tree.   These forwarded modules are specially marked in the dependency tree with a small state image next to their accompanying image (see below).

While processing the dependency tree, Dependency Walker 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, such as an x86 module with an Alpha module.   It scans import and export function tables looking for unresolved external functions.   It checks for circular dependencies, which are allowed, and for circular forwarded dependencies, which are not allowed.   Any errors that are encountered while processing the tree will be displayed using a special image (see below) for the particular modules in error and/or by a message box.

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 contents of the Module Dependency Tree View can also be saved to a text file using the Save Command or Save As Command.

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:

Normal Images
Normal 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.
Dynamic module.   This module was detected during profiling and was dynamically loaded or used by its parent module.   If the module has no parent, then Dependency Walker was unable to determine who loaded the module.   See Using Application Profiling to Detect Dynamic Dependencies for more information.
This module was dynamically loaded by a call to the LoadLibraryEx function with the DONT_RESOLVE_DLL_REFERENCES flag and/or the LOAD_LIBRARY_AS_DATAFILE flag.   These flags cause the module to get mapped into memory without loading its dependent modules or calling the module's DllMain function.
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 Search Order Command for more information.
Invalid module.   See the Module List 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.

▨▨ Dynamic module warning. This module is either missing one or more export functions that the parent module attempted to retrieve using GetProcAddress, 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 functions that the parent module could not locate. For dynamic dependencies, this is usually just a warning, since it is perfectly valid for a module to dynamically check for the existence of a function in another module, even if the function does not exist. If the module failed to load or initialized, then check the Log View for details on the failure.

See the How to Interpret Warnings and Errors in Dependency Walker section for more details on module errors.

It is possible for a module to show up more than once as a dependency of a single parent module. Dependency Walker 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. During runtime profiling, if module A dynamically loads module B, a second instance of module B will appear under module A, but this time with a different image (see above) signifying that it was dynamically loaded. The functions listed in the Parent Import Function List View for this second instance of module B are what module A looked for in module B at runtime using the GetProcAddress function call.

Module 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 <u>Full Paths</u> option.   You may also copy the selected modules' file names or paths to the clipboard by selecting the <u>Copy Command</u>.   The actual text copied will differ depending on how the <u>Full Paths</u> option is set.   If more than one module is selected, a list will be copied to the clipboard with carriage returns after each module.   The complete contents of the Module List View can also be saved to a text file or comma separated value (CSV) file using the <u>Save Command</u> or <u>Save As Command</u>.

There is **not** a one-to-one relationship between the modules listed in this list view and the modules listed in the <u>Module Dependency Tree View</u>.   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.   Since there is not a one-to-one relationship between the two views, the module list view tries to use images for modules that encapsulate the state of all instances of each module in the tree view.   For example, if KERNEL32.DLL appears in the tree view ten times with no errors and one time with an import / export mismatch error, then the list view will show KERNEL32.DLL as having an import / export mismatch error.

Dependency Walker also gives precedence to certain types of dependencies.   If a module is implicitly required for an application to load, then it will appear with the implicit module image in the module list view.   This is true even if the module is also listed as a delay-load or dynamic dependency in the tree view, since an implicit dependency is the most significant type of dependency and is required for the application to load.   If a module is dynamically loaded or is a child of a dynamically loaded module, then it will appear with the dynamic module image in the list view.   If a module is delay-loaded or is a child of a delay-loaded module, then it will appear with the delay-load module image in the list view.   If a module is both a delay-load and dynamic dependency, it will be shown as a dynamic dependency in the list view since modules that actually get dynamically loaded are given precedence over delay-loaded modules that don't get loaded.   This can cause images in the list view to change from delay-load to dynamic as

modules get loaded dynamically.

The following is a table of the primary images that can accompany each module in the Module List View.   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:

Normal Images

All instances of this module were normal and had no errors.   If no delay-load image (hour glass) or dynamic image (star / asterisk) is to the left of this module image, then at least one instance of this module is implicitly required for the root module to load.

All instances of this module are marked as delay-load or are children of modules marked as delay-load.   Modules with this image will change to dynamic dependencies at runtime if the module is actually loaded.

All instances of this module were dynamically loaded and detected during profiling.   See <u>Using Application Profiling to Detect Dynamic Dependencies</u> for more information.

All instances of this module were dynamically loaded by calls to the LoadLibraryEx function with the DONT_RESOLVE_DLL_REFERENCES flag and/or the LOAD_LIBRARY_AS_DATAFILE flag.   These flags cause the module to get mapped into memory without loading its dependent modules or calling the module's DllMain function.   If a module with this image is later loaded without the DONT_RESOLVE_DLL_REFERENCES and LOAD_LIBRARY_AS_DATAFILE flags, then the image will change to the standard dynamic dependency image above.

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 <u>Configure Search Order Command</u> for more information.

Invalid module.   This module will be accompanied by an error message to describe the problem.

Module warning.   At least one instance of 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 load at runtime.   Locate the offending module(s) in the <u>Module Dependency Tree View</u> and then look in the <u>Parent Import Function List View</u> for that module to see the actual unresolved functions that are causing the problem.   This may or may not be an error.   If the offending module(s) are marked as dynamic, then this is just a warning since it is valid for modules to call GetProcAddress to dynamically check for a function and fail to find it.   If the offending module(s) are delay-load, then this is also probably not an error since one reason developers use delay-load dependencies is when they are unsure if a function exists in a dependent module.   If the offending module(s) are implicit or forwarded dependencies,

then this is an error and will cause the parent of those modules to fail to load.   If no export functions are missing, then check the Log View to see if the module error is related to a load failure.

See the How to Interpret Warnings and Errors in Dependency Walker section for more details on module errors.

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

- **Image:** See above list for descriptions.
- **Module:** Full path or file name for the module file.   See the Full Paths option 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.
- **Link Time Stamp:** Date and time that the module was built.   This is a value that the linker stores in the file itself.
- **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 Dependency Walker 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, Intel 64, Alpha AXP, Alpha 64, PowerPC, MIPS R3000 BE (big endian), MIPS R3000, MIPS R4000, MIPS R10000, MIPS WinCE V2, SH3, SH3E, SH4, SH5, ARM, Thumb, MIPS 16, MIPS FPU, MIPS FPU 16, CEE, and CEF.   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 Native, GUI, Console, Win9x driver, OS/2 console, Posix console, WinCE 1.x GUI, and WinCE 2.0+ GUI, EFI, and Xbox.   This value is set by using the linker's /SUBSYSTEM command line option.
- **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.   If one or more of the debug blocks are invalid, then the word "Invalid" will also appear.   This usually means that

debug symbols have been striped from the file, but the debug entries were left behind.

- 🔲 **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.
- 🔲 **Actual Base:** The actual base load address of the module. This value will read "Unknown" until the module has actually been loaded into memory by Dependency Walker's profiler. See the Start Profiling Command for more information. This value will be shown in red if it does not match the preferred base address for the module. Your application will suffer a load-time performance hit for every module that does not load at its preferred base address. This value will read "Data file" if the file was loaded as a data file via a call to LoadLibraryEx with the LOAD_LIBRARY_AS_DATAFILE flag.
- 🔲 **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.
- 🔲 **Load Order:** The load order of the module with respect to other modules. This value will read "Not Loaded" until the module has actually been loaded into memory by Dependency Walker's profiler. See the Start Profiling Command for more information.
- 🔲 **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.

If a module was not found or was not a valid 32-bit or 64-bit Windows binary, then an error message will be displayed in place of the normal column

information for that module.

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.

Dependency Walker 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.   The complete contents of the Parent Import Function List View can also be saved to a text file using the Save Command or Save As Command.

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

     Resolved C import.
     Resolved C++ import.   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.

O#   Resolved ordinal import.

C   Resolved dynamic C import (similar images also exist for C++ and ordinal functions).   The parent module of this module called the GetProcAddress function to dynamically get the address of this function.   This does not necessarily mean the parent module actually used the function address to call the function.

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.

C   Unresolved dynamic C function (similar images also exist for C++ and ordinal functions).   The parent module of this module called the GetProcAddress function to dynamically get the address of this function, but the current module does not export the function.   This is not necessarily an error since one of the reasons modules call GetProcAddress is to see if a function exists in a module.

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.

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.   For dynamic dependencies, this Entry Point field displays the address returned by the GetProcAddress function call.

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 text in the currently sorted column by simply typing in the first few characters of the item you wish to find.   For ordinal and hint values, you may enter decimal or hex (prefaced by 0x) values to search for.

Export Function List View

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

Dependency Walker uses the exported list to check for unresolved external errors in the selected module.   For more information, read the <u>Parent Import Function List View</u> section.

While Dependency Walker 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.   Dependency Walker, like the operating system's loader, also loads the forwarded module if necessary.

C++ functions can be displayed in their native decorated format or in a human readable undecorated format.   See the <u>Undecorate C++ Functions Command</u> for more information.   You may also copy the selected function names to the clipboard by selecting the <u>Copy Command</u>.   The actual text copied will differ depending on how the <u>Undecorate C++ Functions</u> option is set.   If more than one function is selected, a list will be copied to the clipboard with carriage returns after each function.   The complete contents of the Export Function List View can also be saved to a text file using the <u>Save Command</u> or <u>Save As Command</u>.

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

C   C export function that resides in the selected module.
C++   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 the <u>Undecorate C++ Functions Command</u> for more information.
O#   Ordinal export function that resides in the selected module.
C   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   C export function that is called by the selected module in the <u>Module Dependency Tree View</u> (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 <u>Parent Import Function List View</u>.   You can

use the <u>Highlight Matching Item</u> 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.

The Export Function View is comprised of four 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 <u>Undecorate C++ Functions Command</u> 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 <u>Module List View</u> 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 text in the currently sorted column by simply typing in the first few characters of the item you wish to find.   For ordinal and hint values, you may enter decimal or hex (prefaced by 0x) values to search for.

Log View

This view is used to log module warnings, module errors, and all activity while profiling the application for the current Module Session.   For more information on profiling, see the Start Profiling Command, the Using Application Profiling to Detect Dynamic Dependencies section, and the Profile Module Dialog.

While profiling an application, Dependency Walker gathers information from the running process.   The various types of information that can be logged include:

■   The start of the new process.   This is always logged.
■   The exiting of the process.   This is always logged.
■   The creation of a thread.   These are only logged if the **Log thread information** box is checked in the Profile Module Dialog.
■   The exiting of a thread.   These are only logged if the **Log thread information** box is checked in the Profile Module Dialog.
■   The loading of a module.   These are always logged.
■   The unloading of a module.   These are always logged.
■   Any debug output text that the process generates.   These are only logged if the **Log debug output** box is checked in the Profile Module Dialog. Debug output text is logged with a grayed-out color to distinguish it from normal log text.
■   Any first chance exceptions that occur in the process.   These are only logged if the **Log first chance exceptions** box is checked in the Profile Module Dialog.
■   Any second chance exceptions that occur in the process.   These are always logged.   These lines will be colored red.
■   The calling of a module's DllMain function.   These are only logged if either of the two **Log DllMain calls** boxes are checked in the Profile Module Dialog.
■   The return from a module's DllMain function.   These are only logged if either of the two **Log DllMain calls** boxes are checked in the Profile Module Dialog.   This line of log will be shown in red if the DllMain function was called with the DLL_PROCESS_ATTACH message and it returned 0.   If a module returns 0 from its DllMain function while processing the DLL_PROCESS_ATTACH message, then the OS will unload the module and return a failure.   In the case of an implicit dependency, this will cause the entire application to fail to load with an error dialog reading something like "The application failed to initialize properly".   In the case of a dynamic dependency, the call to LoadLibrary will fail with error 1114 (ERROR_DLL_INIT_FAILED), but the application may continue to run.
■   The calling of a LoadLibrary type function.   These are only logged if the **Log LoadLibrary function calls** box is checked in the Profile Module

Dialog.

■   The return from a call to a LoadLibrary type function.   These are only logged if the **Log LoadLibrary function calls** box is checked in the <u>Profile Module Dialog</u>.   This line of log will be colored red if the function fails.

■   Any calls to the GetProcAddress function.   These are only logged if the **Log GetProcAddress function calls** box is checked in the <u>Profile Module Dialog</u>.   This line of log will be colored red if the function fails.

If the **Log a time stamp with each line of log** box is checked in the <u>Profile Module Dialog</u>, then each line of log in the Log View will begin with a time stamp.   Each time stamp shows the number of hours, minutes, seconds, and milliseconds that have elapsed since the process started.   It is important to note that Dependency Walker can significantly impact the performance of certain operations within the application being profiled.   For this reason, these time stamps should probably not be used as an accurate method of measuring the performance of your application.

You may copy text from the Log View using the <u>Copy Command</u>.   The contents of the window can also be saved to a text file using the <u>Save Command</u> or <u>Save As Command</u>.   You can also search the Log View for text using the   <u>Find Command</u> and <u>Find Next Command</u>.

File Menu Commands

The File menu offers the following commands:

<u>Open...</u>	Opens and processes a module file.
<u>Close</u>	Closes the active <u>Module Session Window</u>.
<u>Save</u>	Saves the active <u>Module Session Window</u>.
<u>Save As...</u>	Saves the active <u>Module Session Window</u> with a new name or type.
<u>File 1, 2, 3, ...</u>	Opens and processes the specified module file.
<u>Exit</u>	Exits Dependency Walker.

Edit Menu Commands

The Edit menu offers the following commands:

Copy	Copies the selection in the current view to the clipboard as text.
Select All	Selects all items in the current view.
Find...	Finds text in the Log View.
Find Next	Repeats last find operation in the Log View.
Clear Log Window	Clears the contents of the Log View in the active Module Session Window.

View Menu Commands

The View menu offers the following commands:

System Information...	Displays information about the system.
Expand All	Expands all nodes in the Module Dependency Tree View.
Collapse All	Collapses all nodes in the Module Dependency Tree View.
Auto Expand	When checked, the Module Dependency Tree View will automatically expand to show modules as they are added.
Full Paths	Shows or hides full file paths in the Module Dependency Tree View and the Module List View.
Undecorate C++ Functions	Display undecorated C++ functions names in both the Parent Import Function List View and the Export Function List View.
Highlight Matching Item	Highlights the matching item in the related view.
Highlight Original Instance In Tree	Highlights the original instance of the selected module in the Module Dependency Tree View.
Highlight Previous Instance In Tree	Highlights the previous instance of the selected module in the Module Dependency Tree View.
Highlight Next Instance In Tree	Highlights the next instance of the selected module in the Module Dependency Tree View.
Refresh	Updates all views for the active Module Session Window.
View Module in External Viewer	Opens the selected modules in the external module viewer.
Lookup Function in External Help	Lookup the selected function in the external help collection.
Properties...	Displays the Windows Properties dialog for the selected modules.
Toolbar	Shows or hides the toolbar.
Status Bar	Shows or hides the status bar.

Options Menu Commands

The Options menu offers the following commands:

<u>Configure Module Search Order...</u>	Configure or view the search order used when locating dependent modules.
<u>Configure External Module Viewer...</u>	Configures the external module viewer.
<u>Configure External Function Help Collection...</u>	Configures the external function help collection used to lookup functions.
<u>Configure Handled File Extensions...</u>	Configures what file extensions Dependency Walker handles.

Profile Menu Commands

The Profile menu offers the following commands:

<u>Start Profiling</u>	Executes the module and profiles it for runtime dependencies.
<u>Stop Profiling</u>	Stops execution and profiling of the process.

Window Menu Commands

The Window menu offers the following commands:

<u>Cascade</u>	Arranges windows in an overlapped fashion.
<u>Tile Horizontally</u>	Arranges windows in non-overlapped horizontal tiles.
<u>Tile Vertically</u>	Arranges windows in non-overlapped vertical tiles.
<u>Arrange Icons</u>	Arranges the icons of all minimized windows.
<u>Window 1, 2, 3, …</u>	Activates the specified window.

Help Menu Commands

The Help menu offers the following commands, which provide you assistance with this application:

Help Topics   Displays the table of contents for the online help documentation.

About Dependency Walker...   Displays program information, version, and copyright.

Open Command (File Menu)

The Open Command will display the File Open Dialog, which allows you to open and process a module, or to open a Dependency Walker Image (DWI) file.

You may also open modules directory from an Explorer window by right-clicking on the module you wish to open and choosing "View Dependencies" from the context menu.   In order for this to work, you must tell Dependency Walker what file extensions to handle by using the Handled File Extensions Command.

Dependency Walker uses a multiple document interface that allows more than one Module Session Window to be opened and visible at once.   Use the Window Menu to switch between the multiple open Module Session Windows. See the Window 1, 2, 3, ... Command for more information.

Shortcuts

Keys: CTRL+O

Shell: Drag and drop modules on top of Dependency Walker to open them.

Shell   Right-click on a module file in the Shell and choose "View Dependencies" from the Shell's context menu.

Toolbar:

Close Command (File Menu)

Use this command to close the active Module Session Window.

Shortcuts

Keys: CTRL+F4

Mouse: Single-click on the Close button in the Title Bar of the window you wish to close.

Mouse: Double-click on the System Menu icon in the Title Bar of the window you wish to close.

Save Command (File Menu)

Use this command to save the active Module Session using the same name and type that you have previously saved the file with.   If you have not previously saved the Module Session, then this command behaves just like the Save As Command, which will display the 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 Dependency Walker Image (DWI) file, a comma separated value (CSV) file, or various formats of text files.

Shortcuts

Keys: CTRL+S

Toolbar:

Save As Command (File Menu)

Use this command to display the File Save Dialog, which allows you to save the active Module Session with a new name or type.   Within the File Save Dialog, you can choose to save the file as a Dependency Walker Image (DWI) file, a comma separated value (CSV) file, or various formats of text files.

The Save As Command always displays the File Save Dialog, even if you have previously saved the Module Session 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 active Module Session 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.

1, 2, 3, … Command (<u>File Menu</u>)

Dependency Walker stores the eight most recently opened modules at the bottom of the <u>File menu</u> 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.

Exit Command (File Menu)

Use this command to close all <u>Module Session Windows</u> and exit Dependency Walker.

Shortcuts

Keys: ALT+F4

Mouse: Single-click on the main window's Close button in the <u>Title Bar</u>.

Mouse   Double-click on the main window's System Menu icon in the <u>Title Bar</u>.

Copy Command (Edit Menu)

Use this command to copy the current selection to the clipboard as text. This command is unavailable if there is nothing selected that can be copied. Copying data to the clipboard replaces any contents previously stored on the clipboard.

For the Module Dependency Tree View and the Module List View, the selected module names are copied.   If the Full Paths option is enabled, then complete path strings will be copied, otherwise just the module file names are copied.

For the Parent Import Function List View and the Export Function List View, the selected function names are copied.   If the Undecorate C++ Functions option is enabled, then the undecorated names for C++ functions will be copied, otherwise just the native decorated names are copied.

For the Log View, all highlighted text is copied.

Shortcuts

Keys: CTRL+C

Keys: CTRL+INSERT

Toolbar:

Select All Command (Edit Menu)

Use this command to select all the items in a particular view.   This command only works in the Module List View, the Parent Import Function List View, the Export Function List View, and the Log View.   Select All is often useful before performing a Copy if you wish to copy the entire contents of a view.

Shortcuts

Keys: CTRL+A

Find Command (Edit Menu)

This command will display the Find Dialog, which allows you to search for text in the Log View.

Shortcuts

Keys: CTRL+F

Find Next Command (Edit Menu)

Use this command to repeat the last find operation in the Log View.   If there is no previous find operation, then this command works just like the Find Command, which will display the Find Dialog.

Shortcuts

Keys: F3

Clear Log Window Command (Edit Menu)

Use this command to clear the contents of the Log View.

System Information Command (View Menu)

This command will display the System Information Dialog, which displays detailed information about the operating system.   If the active Module Session is a loaded Dependency Walker Image (DWI) file, then the System Information Dialog will show the system information for the system that the DWI file was saved on.   Otherwise, the System Information Dialog shows information about the current system.

Shortcuts

Toolbar:

Expand All Command (View Menu)

This command will expand all the module nodes in the Module Dependency Tree View, making the entire tree visible.

Shortcuts

Keys: CTRL+E

Collapse All Command (View Menu)

This command will collapse all the module nodes in the Module Dependency Tree View, leaving only the root modules visible.

Shortcuts

Keys: CTRL+W

Auto Expand (<u>View Menu)</u>

When this option is turned on, the <u>Module Dependency Tree View</u> will automatically expand the tree to show modules as they are added.   This includes all modules that are detected during the initial loading of a session, as well as all modules found during profiling.

When this option is turned off, the tree is never automatically expanded as the result of a new module being added.   The only exceptions are the root module and modules that contain errors.   The root module will be expanded to show the immediate dependencies of that root module.   The tree will also be expanded to show any modules that contain errors.   All other branches of the module tree will remain collapsed unless you expand them.

This command can also be used to quickly show all modules that contain errors.   Whenever this option is turned off, the tree will automatically collapse all nodes except for those that contain modules with errors.   If this option is already turned off, you can simply turn it on and back off to force this effect to occur.

Shortcuts

Keys: F8

<u>Toolbar</u>:

Full Paths Command (<u>View Menu</u>)

Use this command to toggle the Full Paths option on or off.   When this option is on, a check mark appears next to the Full Paths menu item and the Full Paths toolbar button is displayed as depressed.

When the Full Paths option is on, both the <u>Module Dependency Tree View</u> and the <u>Module List View</u> 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 <u>Copy Command</u> and <u>Save Command</u> work. When the Full Paths option is on, the <u>Copy Command</u> will copy the full paths of the selected files to the clipboard, otherwise it just copies the file names. For the <u>Save Command</u>, text files and comma separated value (CSV) files will contain full paths when the Full Paths option is on and just file names when it is off.

Shortcuts

Keys: F9

<u>Toolbar</u>: c:\

Undecorate C++ Functions Command (<u>View Menu</u>)

Use this command to toggle the Undecorate C++ Functions option on or off. When this option is on, a check mark appears next to the Undecorate C++ Functions menu item and the Undecorate C++ Functions toolbar button is displayed as depressed.

This option requires that you have IMAGEHLP.DLL on your system.   If this DLL is not found, then the Undecorate C++ Functions option will be disabled. IMAGEHLP.DLL is installed with Windows NT/2000/XP and Windows 95 OSR2 and beyond.

When the Undecorate C++ Functions option is on, both the <u>Parent Import Function List View</u> and the <u>Export Function List View</u> will undecorate C++ functions into human readable function prototypes containing parameter and return types.   When the Undecorate C++ Functions option is off, these views will show C++ functions in their true decorated form.   Dependency Walker can only undecorate functions that use the Microsoft decoration rules.

This option also effects how the <u>Copy Command</u> and <u>Save Command</u> work. When the Undecorate C++ Functions option is on, the <u>Copy Command</u> will copy the undecorated names for C++ functions to the clipboard, otherwise it just copies the true decorated names.   For the <u>Save Command</u>, text files will contain undecorated names for C++ functions when the Undecorate C++ Functions option is on and the true decorated names when it is off.

Shortcuts

Keys: F10

<u>Toolbar</u>:

Highlight Matching Item (<u>View Menu</u>)

This command behaves differently depending on what view has the focus.

If the <u>Module Dependency Tree View</u> has the focus and a module is selected in it, then this command will find that selected module in the <u>Module List View</u> and highlight it.

If the <u>Module List View</u> has the focus and a module is selected in it, then this command will find that selected module in the <u>Module Dependency Tree View</u> and highlight it.

If the <u>Parent Import Function List View</u> has the focus and a function is selected in it, then this command will find the matching function in the <u>Export Function List View</u> and highlight it.   This command will be disabled if the function is unresolved and cannot be found in the <u>Export Function List View</u>.

If the <u>Export Function List View</u> has the focus and a function is selected in it, then this command will find the matching function in the <u>Parent Import Function List View</u> and highlight it.   This command will be disabled if the function is not called by the parent module and cannot be found in the <u>Parent Import Function List View</u>.

Shortcuts

Keys: CTRL+M

Highlight Original Instance In Tree (View Menu)

This command is used to locate the original instance of a module in the Module Dependency Tree View.   It is only enabled when a duplicate module is highlighted.   Duplicate modules are shown with a small arrow in their image.   This command will move the current selection to the original instance of the module.

Shortcuts

Keys: CTRL+K

Highlight Previous Instance In Tree (<u>View Menu</u>)

This command is used to locate the previous instance of the selected module in the <u>Module Dependency Tree View</u>.   It is only enabled when there is a previous instance of the selected module.   This command will move the current selection to the previous instance of the module.

Shortcuts

Keys: CTRL+B

Highlight Next Instance In Tree (<u>View Menu</u>)

This command is used to locate the next instance of the selected module in the <u>Module Dependency Tree View</u>.   It is only enabled when there is a next instance of the selected module.   This command will move the current selection to the next instance of the module.

Shortcuts

Keys: CTRL+N

Refresh Command (<u>View Menu</u>)

This command will force the active <u>Module Session Window</u> to clear all of its views and reprocess the original module.   This can be useful during trouble shooting a module to determine if some action you performed, such as locating and copying a missing module, has alleviated a problem.

Shortcuts

Keys: F5

View Module in External Viewer Command (View Menu)

The external viewer command is provided as a means to launch a secondary module viewer.   The external viewer application is completely user configurable.   See the Configure External Module Viewer... command for more information.

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 Dependency Walker will launch a separate instance of the external viewer application for every module that is selected in the list.

Shortcuts

Keys: ENTER (while one or more modules are highlighted in the active view)

Mouse: Double-click on a module.

Toolbar:

Lookup Function in External Help Command (<u>View Menu</u>)

This command will attempt to find help about the currently selected function by using a help collection installed on your computer or by using the MSDN online collection via the internet.   This command is available when either the <u>Parent Import Function List View</u> or the <u>Export Function List View</u> is active and a named function is highlighted.   To configure what help collection to use for lookups, see the <u>Configure External Function Help Collection</u> command.

Shortcuts

Keys: ENTER (while a function is highlighted in the active view)

Mouse: Double-click on a function.

Properties Command (<u>View Menu</u>)

The properties command is provided as a means to launch the Windows "Properties" dialog for selected modules.

If the active view is the <u>Module Dependency Tree View</u>, <u>Parent Import Function List View</u>, or <u>Export Function List View</u>, then the Properties dialog will be displayed for the module that is currently selected in the <u>Module Dependency Tree View</u>.   If the <u>Module List View</u> has the focus, then Dependency Walker will display a separate Properties dialog for every module that is selected in the list.

Shortcuts

Keys: ALT+ENTER

<u>Toolbar</u>:

Toolbar Command (<u>View Menu</u>)

Use this command to display and hide the Toolbar, which includes buttons for some of the most common commands in Dependency Walker, such as the File Open.   A check mark appears next to the menu item when the Toolbar is displayed.

See <u>Toolbar</u> for more help on using the toolbar.

Status Bar Command (View Menu)

Use this command to display and hide the Status Bar.   A check mark appears next to the menu item when the Status Bar is displayed.

See Status Bar for more help on using the status bar.

Configure Module Search Order Command (<u>Options Menu</u>)

This command will display the <u>Module Search Order Dialog</u>, which allows you to control how Dependency Walker searches your system for dependent files.

Shortcuts

<u>Toolbar</u>:

Configure External Module Viewer Command (Options Menu)

This command will display the Configure External Module Viewer Dialog, which allows you to configure the external viewer application and arguments.

Configure External Function Help Collection Command (Options Menu)

This command will display the Configure External Function Help Collection Dialog, which allows you to configure which help collection to use when the Lookup Function in External Help command is invoked.

Configure Handled File Extensions Command (Options Menu)

This command will display the Handled File Extensions Dialog, which allows you to configure which file extensions Dependency Walker should handle. You can open "handled" files in any explorer window by right-clicking on a file and choosing "View Dependencies" from the context menu.   Handled files also show up in Dependency Walker's File Open Dialog by default.

Start Profiling Command (Profile Menu)

This command will display the Profile Module Dialog, which allows you to configure and start profiling of the active Module Session.

This command will be disabled if any of the following apply:

- You have not loaded any modules into Dependency Walker.
- The application is already being profiled.   If this is the case, then the Stop Profiling Command will be enabled.
- The Module Session represents a loaded Dependency Walker Image (DWI) file.   DWI files are snapshots from a previous time and possibly from a different system.   The files displayed may or may not correspond to files on your current system, and therefore cannot be profiled reliably.
- The root module of the active Module Session does not match the system you are running on.   For example, a 64-bit Alpha module cannot be profiled on a 32-bit x86 computer.
- The root module cannot be a DLL, OCX, or similar type module.   It must be the main executable file (usually ends with .EXE) of an application.

Shortcuts

Keys: F7

Toolbar:

Stop Profiling Command (Profile Menu)

This command will stop profiling the application for the active Module Session.   It will forcefully terminate your application, so it should only be used in situations where the application is not responding to normal methods of closing.   The Stop Profiling Command is only enabled when you are currently profiling the application.   See the Start Profiling Command for more information on profiling your application.

Shortcuts

Keys: SHIFT+F7

Toolbar:

Cascade Command (Window Menu)

Use this command to arrange all non-minimized Module Session Windows in an overlapped fashion.

Shortcuts

Toolbar:

Tile Horizontal Command (Window Menu)

Use this command to arrange all non-minimized Module Session Windows as non-overlapping horizontal tiles.

Shortcuts

Toolbar:

Tile Vertical Command (Window Menu)

Use this command to arrange all non-minimized Module Session Windows as non-overlapping vertical tiles.

Shortcuts

Toolbar:

Arrange Icons Command (<u>Window Menu</u>)

Use this command to arrange the icons for minimized windows at the bottom of the Dependency Walker's main window.

1, 2, 3, ... Command (<u>Window Menu</u>)

Dependency Walker displays a list of currently open <u>Module Session Windows</u> at the bottom of the Window menu.   A check mark appears in front of the <u>Module Session Window</u> name of the active <u>Module Session Window</u>. Choose a module session from this list to make its window active.

Help Topics Command (Help Menu)

Use this command to display the opening screen of Help.   From the opening screen, you can jump to any area of Dependency Walker's online help documentation.

Once you open Help, you can click the Contents button whenever you want to return to the opening screen.

About Dependency Walker Command (Help Menu)

Use this command to display program information, the version, and the copyright of your copy of Dependency Walker.

Context Help Command

Use the Context Help command to obtain help on a particular area of Dependency Walker.   When you choose the Toolbar's Context Help button, the mouse pointer will change to an arrow and question mark.   Then click somewhere in the Dependency Walker window, such as another Toolbar button, menu item, or a view.   The Help topic will be shown for the item you clicked on.

Shortcuts

Keys: SHIFT+F1

Toolbar:

Restore Command (System Menu)

Use this command to return the active window to its size and position before it was Maximized or Minimized.

Shortcuts

Mouse   Single-click the Restore button on the <u>Title Bar</u> of the maximized window or minimize icon bar.

Move Command (System Menu)

Use this command to display the four-headed arrow cursor, which allows you to move the active window or dialog box with the arrow keys.

Note: This command is unavailable if you maximize the window.

Shortcuts

Mouse: Grab the Title Bar of the window and drag the window it to a new location.

Size Command (System Menu)

Use this command to display the four-headed arrow cursor which allows you to re-size the window with the arrow keys.

After the pointer changes to the four-headed arrow:

1. Press one of the DIRECTION keys (left, right, up, or down arrow key) to move the pointer to the border you want to move.   The cursor will change to one of the following images:
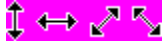
2. Press a DIRECTION key to move the border.
3. Press ENTER when the window is the size you want.

Note: This command is unavailable if you maximize or minimize the window.

Shortcuts

Mouse: Drag the size bars at the corners or edges of the window.

Minimize Command (System Menu)

Use this command to reduce the window to an icon.

Shortcuts

Mouse: Single-click the Minimize button on the <u>Title Bar</u>.

Maximize Command (System Menu)

Use this command to enlarge the active window to fill the available space.

Shortcuts

Mouse: Single-click the Maximize button on the <u>Title Bar</u>.

Mouse: Double-click on the <u>Title Bar</u>.

Close Command (System Menu)

Use this command to close the window.

Shortcuts

Keys: CTRL+F4 to close the active <u>Module Session Window</u>.

Keys: ALT+F4 to close all <u>Module Session Windows</u> and Dependency Walker.

Mouse: Single-click on the Close button in the <u>Title Bar</u> of the window you wish to close.

Mouse: Double-click on the System Menu icon in the <u>Title Bar</u> of the window you wish to close.

Next Window Command (System Menu)

Use this command to switch to the next open Module Session Window. Dependency Walker determines which window is next according to the order in which you opened the Module Session Windows.

See the Previous Window command also.

Shortcuts

Keys: CTRL+F6

Previous Window Command (System Menu)

Use this command to switch to the previous open Module Session Window. Dependency Walker determines which window is previous according to the order in which you opened the Module Session Windows.

See the Next Window command also.

Shortcuts

Keys: SHIFT+CTRL+F6

Next Pane Command

This command allows you to use the keyboard to switch between the different views in a <u>Module Session Window</u>.   The Next Pane Command navigates forward through the views in the following order:

1. <u>Module Dependency Tree View</u>
2. <u>Parent Import Function List View</u>
3. <u>Export Function List View</u>
4. <u>Module List View</u>
5. <u>Log View</u>

See the <u>Previous Pane</u> command for navigating through the views in opposite order.

Shortcuts

Keys: F6

Previous Pane Command

This command allows you to use the keyboard to switch between the different views in a Module Session Window.   The Previous Pane Command navigates backwards through the views in the following order:

1. Log View
2. Module List View
3. Export Function List View
4. Parent Import Function List View
5. Module Dependency Tree View

See the Next Pane command for navigating through the views in opposite order.

Shortcuts

Keys: SHIFT+F6

File Open Dialog

Look in

Lists the available folders and files.   To see how the current folder fits in the hierarchy on your computer, click the down arrow.   To see what's inside a folder, click it.

File and Folder List

This list displays all the files and folders located in the folder specified by the **Look in** field that match the search specifications of the **File name** field and/or the **Files of type** field.   You may select any file in this list and press **Ok** to open the file.   You may also double-click on any file in this list to open the file.

File name

This box allows you to type a full path to a file, a relative path to a file, a path to another folder to browse, a file name to open, or a partial filename with wildcards (* and ?) to search for.   Depending on what you choose to do, the **Look in** field and the **File and Folder List** will update to reflect the change.   If you type an exact match to a particular file, then that file will be opened.

Files of type

Select the types of files you want to open from the drop-down list.   The **File and Folder List** will update to show only the types of files specified by the **Files of type** field.   Dependency Walker provides three options for this list:

> **Handled File Extensions**
>
> Selecting this type will show all files that contain a file extension that you have told Dependency Walker to handle.   To configure what extensions are handled, see the <u>Handled File Extensions Command</u>.   You can load a file with any extension, but this setting only displays the ones that are handled.
>
> **Dependency Walker Image (DWI)**
>
> Selecting this type will show all files with the DWI extension.   DWI files are image files that contain a complete snapshot of a previous <u>Module Session</u>.   By loading a DWI file, you can view the complete results of a previous <u>Module Session</u> without actually being on the system that generated the results.
>
> **All Files (*.*)**
>
> Selecting this option will simply display all files for the current folder.   This can be useful in finding a file that you have not told

dependency walker to handle.

File Save Dialog

Save in

Lists the available folders and files.   To see how the current folder fits in the hierarchy on your computer, click the down arrow.   To see what's inside a folder, click it.

File and Folder List

This list displays all the files and folders located in the folder specified by the **Save in** field that match the search specifications of the **File name** field and/or the **Files of type** field.   You may select any file in this list and press **Ok** to overwrite the file.   You may also double-click on any file in this list to overwrite the file.

File name

This box allows you to type a full path to a file, a relative path to a file, a path to another folder to browse, a file name to save to, or a partial filename with wildcards (* and ?) to search for.   Depending on what you choose to do, the **Save in** field and the **File and Folder List** will update to reflect the change.   If you type in a valid file name and press **Ok** or Enter, then that file will be created and saved to.

Save as type

Select the file format you wish to save the active Module Session to. Dependency Walker provides four options for this list:

Dependency Walker Image (DWI)

DWI files represent a complete snapshot of the current Module Session.   They are binary files that are only recognizable to Dependency Walker.   DWI files may be loaded by Dependency Walker at a future time on any computer to view the complete results of the current Module Session as displayed on the computer that generated the Module Session.

Text (*.txt)

Selecting this option will save the contents of the System Information Dialog, Module Search Order Dialog, Module Dependency Tree View, Module List View, and Log View to a formatted text file that can be viewed with any text viewer.

Text with Import/Export Lists (*.txt)

This option is the same as the **Text** option, but also saves the contents of the Parent Import Function List View and Export Function List View in addition to the contents of the System Information Dialog, Module Search Order Dialog, Module Dependency Tree View,

Module List View, and Log View to a formatted text file that can be viewed with any text viewer.

Comma Separated Values (*.csv)

This option will save the Module List View to a comma separated value (CSV) text file.   CSV files can be easily imported into many applications such as Excel or Access.   They may also be useful with any post processing tools you may write on your own.   Each module in the Module List View uses one line in the CSV file.   The text in each column of the Module List View are separated by commas in the CSV file.   Any text that may contain a comma as part of its text will be put in quotes to prevent the comma from being interpreted as a column separator.

Find Dialog

The following options allow you to search for text in the <u>Log View</u>.

Find what

Fill this field in with the text you wish to locate in the <u>Log View</u>.

Match whole word only

Check this box to limit the search to only finding your text when seen as a whole word and not part of a larger word.   When not checked, all occurrences of your text will be found.   For example, when this option is not checked, searching for "lock" could find words like "clock" and "locker".

Match case

Check this box to limit the search to only finding text that exactly matches the case of your search text.

Find Next

Press this button to look for the next occurrence of your search text.   The search begins from your current cursor location and continues to the end of the view.   For each match that is found, the text will be highlighted in the <u>Log View</u>, and the cursor will be moved to that selection.   You may repeatedly press **Find Next** to continue searching for more matches.

System Information Dialog

This resizable dialog displays information about the current computer, operating system, and user.   If the active <u>Module Session</u> is actually a loaded Dependency Walker Image (DWI) file, then all the information in the System Information Dialog describes the computer that saved the DWI file rather then the current computer.   The caption of this dialog will contain the text "(Local)" if it is displaying live information for the current computer.   For DWI files, the caption will contain the name of the DWI file that the information is stored in.

All the information shown in the System Information Dialog is also saved to text and DWI type files when you use the <u>Save Command</u> or <u>Save As Command</u>.

Close

Closes the dialog.

Refresh

Refreshes the dialog with updated information.   This button will be disabled if the data shown is really from a loaded Dependency Walker Image (DWI) file.

Select All

Selects all the text in the text window.   This button is useful before pressing the **Copy** button if you wish to copy the entire text window.

Copy

Copies the selected text in the text window to the clipboard.   This button is disabled if no text is selected.

Module Search Order Dialog

This resizable dialog is used to configure how Dependency Walker locates dependent modules.   When you first open a module in Dependency Walker, it is scanned for all modules it is dependent on.   Then, all those dependent modules are scanned for their dependent modules.   This recursion is repeated until all modules have been scanned.   Inside each module are various tables that provide this information.   However, only the file names of the dependent files are specified and not complete file paths.   For this reason, it is the job of Dependency Walker to search your system for each file to establish a full path to the files.   This is where the Module Search Order Dialog comes into play.

The Module Search Order Dialog allows you to specify where Dependency Walker should look for dependent modules.   By default, Dependency Walker is set up to simulate the search algorithm that the operating system uses to locate modules.   You can override this default behavior and set up your own custom search criteria.   This can be helpful for various reasons.   For example, maybe you want to check the dependencies of a group of MIPS Windows CE files on your x86 Windows computer.   Since you really don't want Dependency Walker to accidentally pick up x86 Windows modules as dependencies, you can remove all the default search criteria from the search order and just add directories that contain MIPS Windows CE modules.

If the active Module Session is actually a loaded Dependency Walker Image (DWI) file, then the dialog will show the search order that was in use on the computer that created the DWI file.   Also, the caption of the dialog will contain the name of the DWI file, and many of the controls listed below will not be accessible since the search order cannot be modified when viewing the results from a DWI file.   If the current Module Session is not a DWI file, then the dialog's caption will contain the text "(Local)" in it.

The Module Search Order Dialog has seven predefined locations it searches for files.   In addition to these seven locations, you can add search directories of your own.   The seven predefined locations include the following:

Side-by-Side Components (Windows XP only)

Starting with Windows XP, applications can override the operating system's default search order by providing instructions about the versions and/or locations of modules it requires.   These instructions can be stored in a special XML file or as an RT_MANIFEST resource in the main executable itself.

The system's known DLLs list.

These are known modules like KERNEL32.DLL.   When the operating system encounters a known DLL, it skips all rules and loads it from a known place.

The application directory.

This is the directory that the main module of your application lives in.

The 32-bit system directory.

This is your 32-bit system directory.   On Windows NT/2000, it is usually something like C:\WinNT\System32\.   On Windows XP, it is usually something like C:\Windows\System32\.   On Windows 95/98/Me, it is usually something like C:\Windows\System\.

The 16-bit system directory (Windows NT/2000/XP only).

This is your 16-bit Windows directory and only exists on Windows NT/2000/XP.   It is usually located at C:\WinNT\System\.

The system's root OS directory.

This is the directory that your operating system is installed to.   It is usually something like C:\WinNT\ on Windows NT/2000 and C:\Windows\ on Windows 95/98/Me/XP.

The application's registered "App Paths" directories.

This is a set of directories that an application can register for itself in the "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\" section of the registry.   If an application has registered one or more directories, then those directories will be searched for dependent files.   This feature is actually provided by the Shell and not by the core operating system.   When an application is started by calling a Shell function (like ShellExecute or ShellExecuteEx), the Shell checks the registry to see if the application has registered a path in the "App Paths" section.   If so, that path is inserted into the head of the PATH variable for the application about to be started.   Most newer applications use the Shell functions to start other applications, but for applications that call CreateProcess, the applications started will not receive their "App Paths" path as part of their search order.

The system's "PATH" environment variable directories.

The last item in a module's search order is usually the PATH variable. This is a user-definable system variable that is seen by all applications running on a given computer.   It usually contains one or more directories where common modules can be found.

The Module Search Order Dialog has the following controls:

Available Search Groups

If you remove one or more of the predefined search locations from the **Current Search Order** list, they will be added to this list so that you can access them if you wish to add them back to the **Current Search Order**. The locations in this list will not be part of the search order.

Current Search Order

This list displays the current search order.   It can contain any number of the predefined search locations as well as any number of user-defined directories.   When Dependency Walker is searching for a module, it will start at the top of the list and work its way down until the module is found.   If the end of the list is reached with no match, then Dependency Walker gives up and marks the module as "Not Found"

>> (Add)

This moves the highlighted item in the **Available Search Groups** list to the bottom of the **Current Search Order** list.   Once moved, you can move it up the list if necessary using the **Move Up** button.   If no item is highlighted in the **Available Searches** list, then this button will be disabled.

<< (Remove)

This moves the highlighted item in the **Current Search Order** list to the bottom of the **Available Search Groups** list.   If no item is highlighted in the **Current Search Order** list, then this button will be disabled.

Expand

Press this button to show all the files and/or directories that belong to each search group.   When this button is not pressed, just the group names are displayed.

Move Up

This moves the highlighted item in the **Current Search Order** list up one position.   If no item is highlighted, or the first item is highlighted, then this button will be disabled.

Move Down

This moves the highlighted item in the **Current Search Order** list down one position.   If no item is highlighted, or the last item is highlighted, then this button will be disabled.

Load

Press this button to load a Dependency Walker Path (DWP) file from disk. See the Dependency Walker Path (DWP) Files section for more information.   You may also load DWP files from the Command Line when first starting Dependency Walker.

Save

Press this button to save the current search order to a Dependency Walker Path (DWP) file.   See the <u>Dependency Walker Path (DWP) Files</u> section for more information.

Default

This button resets the **Current Search Order** list to its default configuration.   This will cause all user-defined directories to be removed from the list.

Add Directory

This button and text field allow you to add user-defined search directories to the search order.   You can type in a directory you wish to add or press the **Browse** button to graphically pick a directory.   If no text is present in the text field, then the **Add Directory** button will be disabled.   You can add as many user-defined directories as you wish.   Directories are added to the bottom of the **Current Search Order** list.   To move them up the list, use the **Move Up** button.

Browse

This button allows you to graphically choose a directory to be added to the **Current Search Order** list.   Once pushed, a browse dialog will appear allowing you to choose a directory.   If you choose a directory from the browse dialog, it will show up in the **Add Directory** text field.   To actually add the directory to the search order, you need to press the **Add Directory** button.

Configure External Module Viewer Dialog

Command

This field specifies a path to the executable to be run when the <u>View Module in External Viewer</u> command is invoked.   You may use environment variables, like %SystemRoot%, in this path.

Arguments

This field specifies the command line arguments to be passed to the executable specified in the **Command** field when the <u>View Module in External Viewer</u> command is invoked.   You may use a **%1** anywhere in the argument string to represent the full path to the module file.   When the external viewer application is launched, all **%1** tokens will be replaced with the full path to the module file.   You should surround all **%1** arguments in quotes so that the external viewer can handle long filenames with spaces.   For example, "**%1**".   You may also use environment variables in this field.

Browse

This button will display a <u>File Open Dialog</u>, which allows you to browse your system for the executable file to be used as your external viewer.   If a file is chosen in this dialog, the **Command** field will be updated to show the new file.

When you first run Dependency Walker, it defaults to using QUIKVIEW.EXE as your external viewer if you have it on your system.   If it is not found, then it defaults to using DEPENDS.EXE as the external viewer, which will just launch another instance of Dependency Walker.   Here is an example using DUMPBIN.EXE (part of Visual C++) on Windows NT/2000/XP to get header information about a module:

Command: %SystemRoot%\System32\cmd.exe
Arguments: /c dumpbin.exe /headers "%1" > "%TEMP%\headers.txt" & start notepad "%TEMP%\headers.txt"

Configure External Function Help Collection Dialog

This dialog is used to determine what help collection should be used when the <u>Lookup Function in External Help</u> command is invoked.   Dependency Walker will examine your computer and determine what help collections are installed and available for you to use.   It supports collections from MSDN, Visual Studio 6.0, and Visual Studio 7.0.   Dependency Walker can also perform a lookup over the internet using the MSDN online help.   This is useful if you don't have any installed collections, or your collections are out of date.

Use the following MSDN collection

Select this radio button to indicate that you wish to use an installed help collection rather than the online collection.

Collection list

This is a list of help collections that Dependency Walker found installed on your system.   Dependency Walker attempts to sort the list from the most relevant help collection to the least relevant help collection.

Refresh

This button will rescan your system for help collections.

Use MSDN online (Use a %1 to represent the function name)

Select this radio button to indicate that you wish to use the MSDN online help rather than an installed help collection.

URL

This field contains the URL that Dependency Walker will launch in a browser window when you invoke the <u>Lookup Function in External Help</u> command.   Dependency Walker will replace all occurrences of %1 in the URL with the name of the function you are looking up.

Default URL

This button will fill in the URL field with the default URL for using MSDN online.   This default URL was determined at the time Dependency Walker was released and may not work in the future if MSDN online changes the format of their URL.   For this reason, the URL field has been provided so that you can modify the URL to fit your needs.

Handled File Extensions Dialog

This dialog is used to configure what file extensions you wish Dependency Walker to "handle".   Dependency Walker will register itself with your operating system as a viewer for any file extensions you add within this dialog.   Once registered, you can right-click on a handled file in any explorer window and choose "View Dependencies" from the context menu to launch Dependency Walker and process that file.   Handled files are also shown by default in the File Open Dialog when it is first displayed.

You may also use the Handled File Extensions Dialog to remove handled file extensions.   This will remove the "View Dependencies" menu item from the right-click explorer context menu for the extensions you wish to stop handling.

Usually, you will want Dependency Walker to handle all extensions that represent 32-bit or 64-bit Windows modules.   Some common ones are EXE, DLL, and OCX.   However, developers are free to use any extension they wish when creating modules.   Because of this, Dependency Walker provides the option to scan one or more of your disk drives looking for files that are 32-bit or 64-bit Windows modules and automatically add them to your handled file extension list.

Extension

> This field allows you to manually enter an extension and add it to the list. You do not need to enter a period as part of the extension.   After you type in an extension, you need to press the **Add** button to add it to the list.

Add

> This button adds the extension in the **Extension** field to the extension list.   If there is no text in the **Extension** field or the extension entered is already in the list, then this button will be disabled.

Remove

> Removes all the highlighted extensions from the extension list.

Search...

> This button will display the Search for Executable File Extensions Dialog, which allows you to automatically search one or more of your disk drives for 32-bit and 64-bit Windows modules.

Search for Executable File Extensions Dialog

This dialog will automatically search one or more of your disk drives looking for 32-bit and 64-bit Windows modules.   Once the search is complete, you can choose which of the files you want Dependency Walker to handle.

Drives to Search

This list shows all drive letters currently available on your computer.   By default, all drives that are local hard drives are highlighted.   Select the drives you wish to search and press the **Search** button to begin.   While searching, the word "Searching" will appear next to the drive that is currently being searched.

Extensions to Add

Once the searching begins, this list will be populated as 32-bit or 64-bit Windows modules are found.   During the search, the list itself will be disabled, preventing you from unselecting items.   Once the search completes, you can select which files you want Dependency Walker to handle and press the **Add** button.

Search

Once you have selected the drives you wish to search in the **Drives to Search** list, press this button to begin searching.   While searching, all controls in the dialog will be disabled except the **Stop** and **Cancel** buttons.

Stop

This will stop the currently running search.   If there is no currently running search, then this button will be disabled.

Add

Once the search has completed, you can press this button to add all the highlighted extensions in **Extensions to Add** list to the handled list and return to the Handled File Extensions Dialog.

Cancel

Press this button to close the dialog without adding any file extensions to Dependency Walker's handled list.   If a search is currently running, the Cancel button will stop the search first, then close the dialog.

Profile Module Dialog

The profile dialog is used to configure how a module is to be profiled.   It contains the following controls:

Program arguments

> This field can be filled in with any arguments you wish to start the application with.

Starting directory

> This field contains the directory that the application should start in.   By default, this field is filled in with the directory that the main executable lives in.   If you wish to change this directory, you can type in a new directory or press the **Browse** button to graphically choose a new directory.   You can also press the **Default** button to restore this field to its default directory.

Browse...

> This button will display a browse dialog that lets you graphically choose a starting directory for the application.   After you choose a directory, it will appear in the **Starting Directory** field.

Default

> This button will restore the **Starting Directory** field to its default directory.

Clear the log window.

> When this box is checked, the Log View will be cleared before the profile is started.

Simulate ShellExecute by inserting any App Paths directories into the PATH environment variable.

> When this box is checked, Dependency Walker will simulate the ShellExecute function when starting your application.   This ensures that your application's "App Paths" entries are part of the search path.   When this box is not checked, Dependency Walker simply calls CreateProcess to start your application, which does not use the "App Paths" entries. Usually, you should check this box unless you are troubleshooting a problem related to "App Paths" entries.

Log DllMain calls for process attach and process detach messages.

> Dependency Walker monitors all calls to each non-shared module's entrypoint, usually known as the DllMain function.   When this box is checked, all DllMain functions called with the DLL_PROCESS_ATTACH message or DLL_PROCESS_DETACH message will be logged.   If a module returns 0 from its DllMain function while processing the

DLL_PROCESS_ATTACH message, then the OS will unload the module and return a failure.   In the case of an implicit dependency, this will cause the entire application to fail to load with an error dialog reading something like "The application failed to initialize properly".   In the case of a dynamic dependency, the call to LoadLibrary will fail with error 1114 (ERROR_DLL_INIT_FAILED), but the application may continue to run.

Log DllMain calls for all other messages, including thread attach and thread detach.

Dependency Walker monitors all calls to each non-shared module's entrypoint, usually known as the DllMain function.   When this box is checked, all DllMain functions called with the DLL_THREAD_ATTACH message or DLL_THREAD_DETACH message will be logged.

Hook the process to gather more detailed dependency information.

When this item is checked, Dependency Walker will inject a small DLL into the application being profiled to help gather details that can only be gathered from within the application itself.   When the process being profiled is hooked, Dependency Walker is able to track which modules dynamically load other modules at runtime, as well as what functions are dynamically being called into those dynamically loaded modules.   It can also capture the command line arguments passed to child processes.   When a process is not hooked, Dependency Walker can still track all dynamically loaded modules, but cannot provide information about which module loaded the dynamic modules, or what dynamic functions were called.   See the Using Application Profiling to Detect Dynamic Dependencies section for more information.

Log LoadLibrary function calls.

This option is only enabled if the **Hook the process to gather more detailed dependency information** is checked.   When checked, all calls to LoadLibrary type functions will be logged to the Log View.   When not checked, the calls are still processed, but just not displayed in the Log View.

Log GetProcAddress function calls.

This option is only enabled if the **Hook the process to gather more detailed dependency information** is checked.   When checked, all calls to GetProcAddress will be logged to the Log View.   When not checked, the calls are still processed, but just not displayed in the Log View.

Log thread information.

When this option is checked, all thread creations and deletions are logged to the Log View.   Also, all other events logged to the Log View, will have the thread I.D. appended to the end.   This option can be helpful if you are trying to track down what threads are loading modules and calling functions.

Use simple thread numbers instead of actual thread IDs.

This option is only enabled if the **Log thread information** is checked. When checked, simple incrementing numbers are used to represent the different threads rather than true thread I.D.'s, which can be lengthy hexadecimal values.   This makes following a particular thread's activity easier.

Log first chance exceptions.

When this option is checked, all first chance exceptions will be logged to the Log View.   First chance exceptions should be harmless if handled correctly by the application.   Usually, you can leave this option checked, but if you are profiling an application that makes extensive use of first chance exceptions, then you may wish to uncheck this option to reduce unwanted output.   If an application does not handle a first chance exception, then a second chance exception occurs and the application is terminated.   Dependency Walker always logs second chance exceptions, regardless of how this option is set.

Log debug output.

When this option is set, all debug output from the process will be logged to the Log View.

Use full paths when logging file names.

This option lets you control how file names are logged to the Log View. Several of the events that are logged will need to display file names. When this option is checked, full paths to the files will be logged.   When this option is not checked, only the file names will be displayed.

Log a time stamp with each line of log.

When this option is set, each line of log will begin with a time stamp. Each time stamp shows the number of hours, minutes, seconds, and milliseconds that have elapsed since the process started.   It is important to note that Dependency Walker can significantly impact the performance of certain operations within the application being profiled.   For this reason, these time stamps should probably not be used as an accurate method of measuring the performance of your application.

Automatically open and profile child processes.

When this option is checked, Dependency Walker will automatically open and process any child processes of a process being profiled.   For example, if you are profiling application A and it decides to launch application B, then Dependency Walker will open a new Module Session Window for application B and immediately begin to profile it using the same profiling settings as application A.

About Dependency Walker Dialog

This dialog displays program information, the version, and the copyright of your copy of Dependency Walker.

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 Dependency Walker.

There are three ways you can learn what a particular toolbar button's action is.   You can float the mouse over the button and a tool tip will pop up with the command name.   You can press and hold the mouse down over a button and read the text displayed in the Status Bar for a more detailed description. If you do not wish to execute the command, move the mouse off the toolbar button and release the mouse.   Last, you can use the Context Help utility to activate the online help documentation for the toolbar button.

The toolbar can be docked to the top, left, right, and bottom of Dependency Walker's main window, as well as free floated in its own mini window   To change the docking location of the toolbar, simply grab the toolbar along its edge and drag it to where you would like it to go.

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

Click To

■ Opens and processes a module file.   See the Open... command for more information.

■ Saves the current Module Session 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.

■ 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.

■ Configures the search order used when locating dependent modules. See the Configure Search Order command for more information.

▤↓ Starts profiling the current Module Session.   See the Start Profiling Command for more information.

▥↓ Stops profiling the current Module Session.   See the Stop Profiling Command for more information.

▣ Arranges windows in an overlapped fashion.   See the Cascade command for more information.

■ Arranges windows as non-overlapping horizontal tiles.   See the Tile Horizontally command for more information.

▥ Arranges windows as non-overlapping vertical tiles.   See the Tile Vertically command for more information.

■ Enters context help mode.   See the Context Help command for more information.

Status Bar

For Help, press F1

The status bar is displayed at the bottom of Dependency Walker's main window.   To display or hide the status bar, use the Status Bar option from the View menu.

The status bar describes actions of menu items as you use the arrow keys or mouse to navigate through menus.   This area similarly shows messages that describe the actions of Toolbar buttons as you depress them and before releasing them.   If after viewing the description of the toolbar button command you wish not to execute the command, then move the mouse pointer off the toolbar button and release the mouse button.

Title Bar

The title bar is located along the top of a window.   For Dependency Walker's main window (shown above), it contains the name of the application and the active module session name if a module has been loaded.   For a Module Session Window, it will contain the name of the session module.

To move a window, drag the title bar.   To resize a window, drag the size bars at the corners or edges of the window.

Dependency Walker's main window's title bar contains the following elements:
- System Menu button.   This is actually displayed as a small Dependency Walker icon on left size of the Title Bar
- Name of the application, "Dependency Walker"
- Name of the active Module Session; for example, "notepad.exe"
- Minimize button
- Restore/Maximize button
- Close button

Scroll Bars

Scroll bars are displayed at the right and bottom edges of each view.   The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document.   You can use the mouse to scroll to other parts of the view.

No Help Available

Sorry, there is no help available for this area or topic.