NAME

winebuild – Wine dll builder

SYNOPSIS

winebuild [*options*] [*inputfile*...]

DESCRIPTION

winebuild generates the assembly files that are necessary to build a Wine dll, which is basically a Win32 dll encapsulated inside a Unix library.

winebuild has different modes, depending on what kind of file it is asked to generate. The mode is specified by one of the mode options specified below. In addition to the mode option, various other command-line option can be specified, as described in the **OPTIONS** section.

MODE OPTIONS

You have to specify exactly one of the following options, depending on what you want winebuild to generate.

–-dll Build an assembly file from a .spec file (see **SPEC FILE SYNTAX** for details), or from a standard Windows .def file. The .spec/.def file is specified via the **-E** option. The resulting file must be assembled and linked to the other object files to build a working Wine dll. In this mode, the *input files* should be the list of all object files that will be linked into the final dll, to allow **winebuild** to get the list of all undefined symbols that need to be imported from other dlls.

–-exe Build an assembly file for an executable. This is basically the same as the **--dll** mode except that it doesn't require a .spec/.def file as input, since an executable need not export functions. Some executables however do export functions, and for those a .spec/.def file can be specified via the **-E** option. The executable is named from the .spec/.def file name if present, or explicitly through the **-F** option. The resulting file must be assembled and linked to the other object files to build a working Wine executable, and all the other object files must be listed as *input files.*

–-def Build a .def file from a spec file. The .spec file is specified via the **-E** option. This is used when building dlls with a PE (Win32) compiler.

–-implib
Build a .a import library from a spec file. The .spec file is specified via the **-E** option. If the output library name ends in .delay.a, a delayed import library is built.

–-resources
Generate a .o file containing all the input resources. This is useful when building with a PE compiler, since the PE binutils cannot handle multiple resource files as input. For a standard Unix build, the resource files are automatically included when building the spec file, so there's no need for an intermediate .o file.

OPTIONS

–-as-cmd=*as-command*
Specify the command to use to compile assembly files; the default is **as**.

–b, --target=*cpu-manufacturer*[**-***kernel*]**-***os*
Specify the target CPU and platform on which the generated code will be built. The target specification is in the standard autoconf format as returned by config.sub.

–B *directory*
Add the directory to the search path for the various binutils tools like **as**, **nm** and **ld**.

–-cc-cmd=*cc-command*
Specify the C compiler to use to compile assembly files; the default is to instead use the assembler specified with **--as-cmd**.

–d, --delay-lib=*name*
Set the delayed import mode for the specified library, which must be one of the libraries imported with the **-l** option. Delayed mode means that the library won't be loaded until a function imported from it is actually called.

–D *symbol*
Ignored for compatibility with the C compiler.

–e, --entry=*function*
Specify the module entry point function; if not specified, the default is **DllMain** for dlls, and **main** for executables (if the standard C **main** is not defined, **WinMain** is used instead). This is only valid for Win32 modules.

–E, --export=*filename*
Specify a .spec file (see **SPEC FILE SYNTAX** for details), or a standard Windows .def file that defines the exports of the DLL or executable that is being built.

–-external-symbols
Allow linking to external symbols directly from the spec file. Normally symbols exported by a dll have to be defined in the dll itself; this option makes it possible to use symbols defined in another Unix library (for symbols defined in another dll, a *forward* specification must be used instead).

–f *option*
Specify a code generation option. Currently **–fPIC** and **–fasynchronous-unwind-tables** are supported. Other options are ignored for compatibility with the C compiler.

–-fake-module
Create a fake PE module for a dll or exe, instead of the normal assembly or object file. The PE module contains the resources for the module, but no executable code.

–F, --filename=*filename*
Set the file name of the module. The default is to use the base name of the spec file (without any extension).

–h, --help
Display a usage message and exit.

–H, --heap=*size*
Specify the size of the module local heap in bytes (only valid for Win16 modules); default is no local heap.

–I *directory*
Ignored for compatibility with the C compiler.

–k, --kill-at
Remove the stdcall decorations from the symbol names in the generated .def file. Only meaningful in **--def** mode.

–K *flags*
Ignored for compatibility with the C compiler.

–-large-address-aware
Set a flag in the executable to notify the loader that this application supports address spaces larger than 2 gigabytes.

–-ld-cmd=*ld-command*
Specify the command to use to link the object files; the default is **ld**.

–L, --library-path=*directory*
Append the specified directory to the list of directories that are searched for import libraries.

–l, --library=*name*
Import the specified library, looking for a corresponding *libname.def* file in the directories specified with the **-L** option.

–m16, -m32, -m64
Generate respectively 16-bit, 32-bit or 64-bit code.

−marm, −mthumb, −march=*option*, **−mcpu=***option*, **−mfpu=***option*, **−mfloat-abi=***option*
Set code generation options for the assembler.

−munix
Build a library that imports standard functions from the Unix C library instead of the Windows runtime.

−M, --main-module=*module*
When building a 16-bit dll, set the name of its 32-bit counterpart to *module*. This is used to enforce that the load order for the 16-bit dll matches that of the 32-bit one.

−N, --dll-name=*dllname*
Set the internal name of the module. It is only used in Win16 modules. The default is to use the base name of the spec file (without any extension). This is used for KERNEL, since it lives in KRNL386.EXE. It shouldn't be needed otherwise.

−-nm-cmd=*nm-command*
Specify the command to use to get the list of undefined symbols; the default is **nm**.

--nxcompat=*yes*|*no*
Specify whether the module is compatible with no-exec support. The default is yes.

−o, --output=*file*
Set the name of the output file (default is standard output). If the output file name ends in .o, the text output is sent to a temporary file that is then assembled to produce the specified .o file.

−r, --res=*rsrc.res*
Load resources from the specified binary resource file. The *rsrc.res* file can be produced from a source resource file with **wrc**(1) (or with a Windows resource compiler).
This option is only necessary for Win16 resource files, the Win32 ones can simply listed as *input files* and will automatically be handled correctly (though the **−r** option will also work for Win32 files).

--save-temps
Do not delete the various temporary files that **winebuild** generates.

--subsystem=*subsystem*[**:***major*[**.***minor*]]
Set the subsystem of the executable, which can be one of the following:
console for a command line executable,
windows for a graphical executable,
native for a native-mode dll,
wince for a ce dll.
The entry point of a command line executable is a normal C **main** function. A **wmain** function can be used instead if you need the argument array to use Unicode strings. A graphical executable has a **WinMain** entry point.
Optionally a major and minor subsystem version can also be specified; the default subsystem version is 4.0.

−u, --undefined=*symbol*
Add *symbol* to the list of undefined symbols when invoking the linker. This makes it possible to force a specific module of a static library to be included when resolving imports.

−v, --verbose
Display the various subcommands being invoked by **winebuild**.

−-version
Display the program version and exit.

−w, --warnings
Turn on warnings.

SPEC FILE SYNTAX

General syntax

A spec file should contain a list of ordinal declarations. The general syntax is the following:

ordinal functype [*flags*] *exportname* **(** [*args...*] **)** [*handler*]
ordinal **variable** [*flags*] *exportname* **(** [*data...*] **)**
ordinal **extern** [*flags*] *exportname* [*symbolname*]
ordinal **stub** [*flags*] *exportname* [**(***args...***)**]
ordinal **equate** [*flags*] *exportname data*
*comments*

Declarations must fit on a single line, except if the end of line is escaped using a backslash character. The **#** character anywhere in a line causes the rest of the line to be ignored as a comment.

ordinal specifies the ordinal number corresponding to the entry point, or '@' for automatic ordinal allocation (Win32 only).

flags is a series of optional flags, preceded by a '-' character. The supported flags are:

-norelay
The entry point is not displayed in relay debugging traces (Win32 only).

-noname
The entry point will be exported by ordinal instead of by name. The name is still available for importing.

-ret16 The function returns a 16-bit value (Win16 only).

-ret64 The function returns a 64-bit value (Win32 only).

-register
The function uses CPU register to pass arguments.

-private
The function cannot be imported from other dlls, it can only be accessed through GetProcAddress.

-ordinal
The entry point will be imported by ordinal instead of by name. The name is still exported.

-thiscall
The function uses the *thiscall* calling convention (first parameter in %ecx register on i386).

-fastcall
The function uses the *fastcall* calling convention (first two parameters in %ecx/%edx registers on i386).

-import
The function is imported from another module. This can be used instead of a *forward* specification when an application expects to find the function's implementation inside the dll.

-arch=*cpu*[**,***cpu*]
The entry point is only available on the specified CPU architecture(s). The names **win32** and **win64** match all 32-bit or 64-bit CPU architectures respectively. In 16-bit dlls, specifying **-arch=win32** causes the entry point to be exported from the 32-bit wrapper module.

Function ordinals

Syntax:
ordinal functype [*flags*] *exportname* **(** [*args...*] **)** [*handler*]

This declaration defines a function entry point. The prototype defined by *exportname* **(** [*args...*] **)** specifies

the name available for dynamic linking and the format of the arguments. '@' can be used instead of *exportname* for ordinal-only exports.

functype should be one of:

stdcall for a normal Win32 function

pascal for a normal Win16 function

cdecl for a Win16 or Win32 function using the C calling convention

varargs
for a Win16 or Win32 function using the C calling convention with a variable number of arguments

args should be one or several of:

word (16-bit unsigned value)

s_word
(16-bit signed word)

long (pointer-sized integer value)

int64 (64-bit integer value)

int128 (128-bit integer value)

float (32-bit floating point value)

double (64-bit floating point value)

ptr (linear pointer)

str (linear pointer to a null-terminated ASCII string)

wstr (linear pointer to a null-terminated Unicode string)

segptr (segmented pointer)

segstr (segmented pointer to a null-terminated ASCII string).

Note: The 16-bit and segmented pointer types are only valid for Win16 functions.

handler is the name of the actual C function that will implement that entry point in 32-bit mode. The handler can also be specified as *dllname.function* to define a forwarded function (one whose implementation is in another dll). If *handler* is not specified, it is assumed to be identical to *exportname.*

This first example defines an entry point for the 32-bit GetFocus() call:

```
@ stdcall GetFocus() GetFocus
```

This second example defines an entry point for the 16-bit CreateWindow() call (the ordinal 100 is just an example); it also shows how long lines can be split using a backslash:

```
100 pascal CreateWindow(ptr ptr long s_word s_word s_word \
  s_word word word word ptr) WIN_CreateWindow
```

To declare a function using a variable number of arguments, specify the function as **varargs** and declare it in the C file with a '...' parameter for a Win32 function, or with an extra VA_LIST16 argument for a Win16 function. See the wsprintf* functions in user.exe.spec and user32.spec for an example.

Variable ordinals

Syntax:
ordinal **variable** [*flags*] *exportname* **(** [*data...*] **)**

This declaration defines data storage as 32-bit words at the ordinal specified. *exportname* will be the name available for dynamic linking. *data* can be a decimal number or a hex number preceded by "0x". The following example defines the variable VariableA at ordinal 2 and containing 4 ints:

2 variable VariableA(-1 0xff 0 0)

This declaration only works in Win16 spec files. In Win32 you should use **extern** instead (see below).

Extern ordinals

Syntax:
ordinal **extern** [*flags*] *exportname* [*symbolname*]

This declaration defines an entry that simply maps to a C symbol (variable or function). It only works in Win32 spec files. *exportname* will point to the symbol *symbolname* that must be defined in the C code. Alternatively, it can be of the form *dllname***.***symbolname* to define a forwarded symbol (one whose implementation is in another dll). If *symbolname* is not specified, it is assumed to be identical to *exportname.*

Stub ordinals

Syntax:
ordinal **stub** [*flags*] *exportname* [**(***args...***)**]

This declaration defines a stub function. It makes the name and ordinal available for dynamic linking, but will terminate execution with an error message if the function is ever called.

Equate ordinals

Syntax:
ordinal **equate** [*flags*] *exportname data*

This declaration defines an ordinal as an absolute value. *exportname* will be the name available for dynamic linking. *data* can be a decimal number or a hex number preceded by "0x".

AUTHORS

winebuild has been worked on by many people over the years. The main authors are Robert J. Amstadt, Alexandre Julliard, Martin von Loewis, Ulrich Weigand and Eric Youngdale. Many other people have contributed new features and bug fixes. For a complete list, see the git commit logs.

BUGS

It is not yet possible to use a PE-format dll in an import specification; only Wine dlls can be imported.

Bugs can be reported on the **Wine bug tracker** ⟨https://bugs.winehq.org⟩.

AVAILABILITY

winebuild is part of the Wine distribution, which is available through WineHQ, the **Wine development headquarters** ⟨https://www.winehq.org/⟩.

SEE ALSO

wine(1), **winegcc**(1), **wrc**(1),
Wine documentation and support ⟨https://www.winehq.org/help⟩.