NAME

mkbundle, mkbundle2 − Creates a bundled executable.

SYNOPSIS

mkbundle [options] assembly1 [assembly2 ...]

DESCRIPTION

mkbundle generates an executable program that will contain static copies of the assemblies listed on the command line. By default only the assemblies specified in the command line will be included in the bundle. To automatically include all of the dependencies referenced, use the "--deps" command line option.

There are two modes of operation, one uses an existing Mono binary or a server-hosted list of binaries and is enabled when you use either the **--cross, --sdk** or the **--runtime** command line options.

An older mechanism creates a small C stub that links against the libmono library to produce a self-contained executable and requires a C compiler. It is described in the "OLD EMBEDDING" section below.

For example, to create a bundle for hello world, use the following command:

```
$ mkbundle -o hello --simple hello.exe
```

You can configure options to be passed to the Mono runtime directly into your executable, for this, use the *--options* flag. For example, the following disables inlining, by passing the "-O=-inline" command line option to the embedded executable:

```
$ mkbundle -o hello --options -O=-inline --simple hello.exe
```

The simple version allows for cross-compiling, this requires a Mono
runtime to be installed in the ~/.mono/targets/TARGET/mono to be
available. You can use the "--local-targets" to list all available
targets, and the "--cross" argument to specify the target, like this:

```
$ mkbundle --local-targets
Available targets:
        default   - Current System Mono
        4.4.0-macosx-x86
        4.4.0-debian-8-arm64
$ mkbundle --cross 4.4.0-debian-8-powerpc hello.exe -o hello-debian
```

The above will bundle your native library into hello-debian for a Debian 8 system running on a PowerPC machine.

We provide pre-packages binaries for Mono for various architectures, which allow you to cross compile, use the **--list-targets** to get a list of all targets supported, and use the **--fetch-target** flag to retrieve a target that you do not have installed, like this:

```
$ mkbundle --list-targets
Cross-compilation targets available:
4.4.0-linux-libc2.13-amd64
4.4.0-linux-libc2.13-armel
4.4.0-linux-libc2.13-armhf
4.4.0-linux-libc2.13-i386
4.4.0-macos-10.7-amd64
4.4.0-macos-10.7-i386
4.4.2-linux-libc2.13-amd64
4.4.2-linux-libc2.13-armel
4.4.2-linux-libc2.13-armhf
```

```
4.4.2-linux-libc2.13-i386
4.4.2-macos-10.7-amd64
4.4.2-macos-10.7-i386

$ mkbundle --fetch-target 4.4.2-macos-10.7-i386
```

And then you can produce a binary that will run on 32-bit Mono on MacOS:

```
$ mkbundle --cross 4.4.2-macos-10.7-i386 hello.exe -o hello-macos
```

Downloaded targets are stored **~/.mono/targets** directory.

OPTIONS

--config FILE
Specifies that a DLLMAP Mono config file must be bundled as well. In the simple and cross compiler modes, if no config file is specified the one for the current target is picked (either the system one in the case of the simple mode, or the one that came from the cross compilation target for the cross compiling mode).

--config-dir DIR
When passed, DIR will be set for the MONO_CFG_DIR environment variable

--cross target
Use this to request mkbundle generate a cross-compiled binary. It Creates a bundle for the specified target platform. The target must be a directory in ~/.mono/targets/ that contains an SDK installation as produced by the mono-package-runtime tool. You can get a list of the precompiled versions of the runtime using --list-targets and you can fetch a specific target using the --fetch-target command line option.

This flag is mutually exclusive with *--sdk* which is used to specify an absolute path to resolve the Mono runtime from and the --runtime option which is used to manually construct the cross-platform package.

--deps This option will bundle all of the referenced assemblies for the assemblies listed on the command line option. This is useful to distribute a self-contained image.

--env KEY=VALUE
Use this to hardcode an environment variable at runtime for KEY to be mapped to VALUE. This is useful in scenarios where you want to enable certain Mono runtime configuration options that are controlled by environment variables.

--fetch-target target
Downloads a precompiled runtime for the specified target from the Mono distribution site.

--i18n encoding
Specified which encoding tables to ship with the executable. By default, Mono ships the supporting I18N.dll assembly and the I18N.West.dll assembly. If your application will use the System.Text.Encoding.GetEncoding with encodings other than the West encodings, you should specify them here.

You can use the **none** parameter to request that no implicit encodings should be bundled, including the supporting I18N.dll, use this option if you have ran a linker on your own.

You can use the **all** flag to bundle all available encodings.

Or you can use a comma delimited list of the workds CJK, MidWest, Other, Rare and West to specificy which encoding assemblies to distribute.

-L path Adds the 'path' do the search list for assemblies. The rules are the same as for the compiler -lib: or -L flags.

--library [LIB,]PATH
: Embeds the dynamic library file pointed to by ‘PATH’ and optionally give it the name ‘LIB’ into the bundled executable. This is used to ship native library dependencies that are unpacked at startup and loaded from the runtime. Multiple libraries should be specified in dependency order, where later ones on the command line depend on earlier ones.

--lists-targets
: Lists all of the available local cross compilation targets available as precompiled binaries on the Mono distribution server.

--local-targets
: Lists all of the available local cross compilation targets.

--cil-strip PATH
: Provides a CIL stripper that mkbundle will use if able to. The intended use is to help reduce file size on AOT.

--in-tree path/to/mono/source/root
: Provides mkbundle with a mono source repository from which to pull the necessary headers for compilation. This allows mkbundle to run out of the project’s source tree, useful for working with multiple runtimes and for testing without installing.

--managed-linker PATH
: Provides mkbundle access to a managed linker to preprocess the assemblies.

--machine-config FILE
: Uses the given FILE as the machine.config file for the generated application. The machine config contains an XML file that is used by System.Configuration APIs to configure the .NET stack. Typically this is *$prefix/etc/mono/4.5/machine.config.*

  If you want to disable this automatic bundling, you can use the *--no-machine-config* flag. In the simple and cross compiler modes, if no machine.config file is specified the one for the current target is picked (either the system one in the case of the simple mode, or the one that came from the cross compilation target for the cross compiling mode).

--no-config
: In simple or cross compiling mode, this prevents mkbundle from automatically bundling a config file.

--nodeps
: This is the default: *mkbundle* will only include the assemblies that were specified on the command line to reduce the size of the resulting image created.

--no-machine-config
: In simple or cross compiling mode, this prevents mkbundle from automatically bundling a machine.config file.

-o filename
: Places the output on ‘out’. If the flag -c is specified, this is the C host program. If not, this contains the resulting executable.

--options OPTS
: Since the resulting executable will be treated as a standalone program, you can use this option to pass configuration options to the Mono runtime and bake those into the resulting executable. These options are specified as *OPTS.*

  You can use the above to configure options that you would typically pass on the command line to Mono, before the main program is executed.

  Additionally, users of your binary can still configure their own options by setting the *MONO_ENV_OPTIONS* environment variable.

--sdk SDK_PATH

Use this flag to specify a path from which mkbundle will resolve the Mono SDK from. The SDK path should be the prefix path that you used to configure a Mono installation. And would typically contain files lik *SDK_PATH/bin/mono* , *SDK_PATH/lib/mono/4.5* and so on.

When this flag is specified, *mkbundle* will resolve the runtime, the framework libraries, unmanaged resources and configuration files from the files located in this directory.

This flag is mutually exlusive with *--cross*

--target-server SERVER

By default the mkbundle tool will download from a Mono server the target runtimes, you can specify a different server to provide cross-compiled runtimes.

--mono-api-struct-path FILE

FILE points to a file with the definition of the *BundleMonoAPI* structure which contains the required pointers to various Mono API functions used throughout the generated code. This mechanism is meant to be used by third parties which embed the Mono runtime and dynamically load and initialize it as part of the application startup, in which case the Mono APIs will not be available for the shared library loader and the bundle will fail to work (one example of such an embedding third party is Xamarin.Android).

After providing the definition FILE, the embedder must call the *void initialize_mono_api (const BundleMonoAPI *info)* function found in the generated code **before** calling *void mono_mkbundle_init ()*. The structure passed to *initialize_mono_api* doesn't need to be dynamically allocated as its contents is copied to the local structure in the generated code and no pointer to the passed structure is retained or used after *initialize_mono_api* returns.

The list of pointers is not documented here. Instead, please look at the *bundle-mono-api.inc* file found in the mkbundle source directory in your Mono source tree (*mcs/tools/mkbundle*) or in the Mono's GitHub repository, https://github.com/mono/mono/blob/master/mcs/tools/mkbundle/bundle-mono-api.inc

Please note that your structure must match the one expected by your version of the Mono runtime.

The file must also define the *mkbundle_log_error* function with the following signature:

```
static void mkbundle_log_error (const char *format, ...) { }
```

The function should implement logging API specific to the embedder.

OLD EMBEDDING

The old embedding system compiles a small C stub that embeds the C code and compiles the resulting executable using the system compiler. This requires both a working C compiler installation and only works to bundle binaries for the current host.

The feature is still available, but we recommend the simpler, faster and more convenient new mode.

For example, to create a bundle for hello world, use the following command:

```
$ mkbundle -o hello hello.exe
```

The above will pull hello.exe into a native program called "hello". Notice that the produced image still contains the CIL image and no precompilation is done.

In addition, it is possible to control whether *mkbundle* should compile the resulting executable or not with the -c option. This is useful if you want to link additional libraries or control the generated output in more detail. For example, this could be used to link some libraries statically:

```
$ mkbundle -c -o host.c -oo bundles.o --deps hello.exe

$ cc host.c bundles.o /usr/lib/libmono.a -lc -lrt
```

You may also use *mkbundle* to generate a bundle you can use when embedding the Mono runtime in a native application. In that case, use both the -c and --nomain options. The resulting host.c file will not have a main() function. Call mono_mkbundle_init() before initializing the JIT in your code so that the bundled assemblies are available to the embedded runtime.

OLD EMBEDDING OPTIONS

These options can only be used instead of using the **--cross, --runtime** or **--simple** options.

-c Produce the stub file, do not compile the resulting stub.

-oo filename
Specifies the name to be used for the helper object file that contains the bundle.

--keeptemp
By default *mkbundle* will delete the temporary files that it uses to produce the bundle. This option keeps the file around.

--nomain
With the -c option, generate the host stub without a main() function.

--static By default *mkbundle* dynamically links to mono and glib. This option causes it to statically link instead.

-z Compresses the assemblies before embedding. This results in smaller executable files, but increases startup time and requires zlib to be installed on the target system.

AOT Options

These options support an mkbundle using AOT compilation with static linking. A native compiler toolchain is required.

--aot-runtime PATH
Provide the path to the mono runtime to use for AOTing assemblies.

--aot-dedup
(Experimental) Deduplicate AOT'ed methods based on a unique mangling of method names.

--aot-mode MODE
MODE can be either "full" or "llvmonly" at this time. Currently, mkbundle supports three AOT modes. The default mode (this option unset) will AOT methods but will fall back on runtime codegen where it is much faster or offers a more full compatibility profile. The "full" setting will generate the necessary stubs to not require runtime code generation. The "llvmonly" setting does the same, but forces all codegen to go through the llvm backend.

WINDOWS

If you are using the old embedding on Windows systems, it possible to use a Unix-like toolchain like cygwin's and install gcc, gcc-mingw packages or use Visual Studio 2015/2017 VC toolchain together with Clang for Visual Studio as assembler. Clang can be installed as an individual component, "Clang/C2", using Visual Studio installer.

Using Visual Studio toolchain, mkbundle will, by default, use latest installed Visual Studio compiler and linker as well as Windows SDK. If executed from one of the Visual Studio developer command prompts, mkbundle will retrieve information directly from that build environment.

ENVIRONMENT VARIABLES

AS Assembler command. The default is "as". For Visual Studio, default is "clang.exe". If "clang.exe" for Visual Studio is not installed, mkbundle will fall back using "as".

CC C compiler command. The default is "cc" for Linux, "gcc" for cygwin and "cl.exe" for Visual Studio.

MONO_BUNDLED_OPTIONS
Options to be passed to the bundled Mono runtime, separated by spaces. See the mono(1) manual page or run mono --help.

WINDOWS VISUAL STUDIO ENVIRONMENT VARIABLES

VisualStudioVersion
Visual Studio version used in mkbundle build. Default, latest installed Visual Studio version. Values, "14.0" for Visual Studio 2015 or "15.0" for Visual Studio 2017.

WindowsSdkVersion
Windows SDK version used in mkbundle build. Default/unknown, latest installed Windows SDK. Values, "8.1", "10.0.10240.0", "10.0.15063.0" etc.

VSCMD_ARG_TGT_ARCH
Output target architecture used in mkbundle build. Default/unknown, use architecture of .NET runtime executing mkbundle. Values, "x86" or "x64". NOTE, when running from a Visual Studio command prompt, this variable should already be set by the command prompt and match the rest of that build environment.

INCLUDE
Override all custom include paths passed to "cl.exe". Predefined by Visual Studio developer command prompt or auto detected by mkbundle when undefined.

LIB Override all custom library paths passed to "link.exe". Predefined by Visual Studio developer command prompt or auto detected by mkbundle when undefined.

MONOPREFIX
Use a custom Mono SDK install root matching the output target architecture (x86/x64). Default, mkbundle will look for installed Mono SDK's matching targeted architecture.

MONOLIB
Use a different mono library name or an absolute path to the mono library passed to linker. Default, mkbundle will use default mono library name depending on mkbundle dynamic/static use case. NOTE, supplied mono library needs to match mkbundle dynamic/static use case and target architecture.

VCCRT
Override C-runtime library linker settings. Default "MD", mkbundle will use dynamic C-runtime linking on Windows compatible with Mono SDK distribution. If a custom built Mono runtime using static C-Runtime linkage is used, setting this variable to "MT" will link using static C-runtime libraries.

VCSUBSYSTEM
Override Windows subsystem. Default, "windows". If console subsystem is preferred, use "console". NOTE, if console output is expected from output target process then set this variable to "console".

FILES

This program will load referenced assemblies from the Mono assembly cache.

Targets are loaded from ~/.mono/targets/TARGETNAME/mono

BUGS

MAILING LISTS

Visit http://lists.ximian.com/mailman/listinfo/mono-devel-list for details.

WEB SITE

Visit: http://www.mono-project.com for details

SEE ALSO

mcs(1),mono(1),**mono-config(5).**