# BlitzMax Make (bmk)

EDIT

`bmk` is used to build BlitzMax applications and modules. It scans source files for imports and build options. Next to feeding information to `bcc` it is also executing scripts before and after building projects (see "scripts" page, which also explains the automated `pre.bmk` and `post.bmk` scripts).

## Command line syntax

```
bmk <operation> [options] source
```

(Note that you will need `./bmk` on *nix platforms, if you want to run bmk from the `bin` directory.)

### Operations

#### `makeapp`

Builds an application from a single root source file.

#### `makemods`

Builds a set of modules.

#### `makelib`

On Win32, builds a dynamic linked library (DLL) from the source file.

#### `makebootstrap`

Build a bootstrap package. To finish building the transpiled C files on the client then.

Clean the given modules by removing all files in their ".bmx" directories. To delete also other files add the `-k` parameter.

**`zapmod <modulename> <outputfile>`**

Pack a given module into a single file.

**`unzapmod <infile>`**

Unpack a given file into a module directory.

**`ranlibdir <directory>`**

Adds and/or updates object files of static libraries in the given directory.

## Options

**`-a`**

Recompiles all source/modules regardless of timestamp. By default, only those modified since the last build are recompiled.

**`-b <custom appstub module>`**

Builds an app using a custom appstub (The default is brl.appstub).

This can be useful when you want more control over low-level application state and the debugger.

**`-d`**

Builds a debug version. (This is the default for makeapp)

**`-g <architecture>`**

Compiles to the specified architecture. (the default is the native for the current binary - For example, it will be x86 for an x86-built bmk)

- Linux : `x86` , `x64` , `arm` , `arm64`
- iOS : `x86` , `x64` (simulator), `armv7` , `arm64`
- Android : `x86` , `x64` , `arm` , `armeabi` , `armeabiv7a` , `arm64v8a`
- RaspberryPi : `arm` , `arm64`
- NX : `arm64`

**`-gdb`**

Generates line mappings suitable for GDB debugging.

Backtrace (etc.) will show .bmx relative source lines rather than that of the generated code.

**`-h`**

Builds multithreaded version. (By default, the single threaded version is built when used with `bcc` of BlitzMax legacy while the newer BlitzMax NG `bcc` builds multi-threaded versions.)

**`-i`**

Creates a Universal build for supported platforms (Mac OS X and iOS).

Support of this param is only partial. !TODO!

**`-l <target platform>`**

Cross-compiles to the specific target platform.

Valid targets are `win32` , `linux` , `macos` , `ios` , `android` , `raspberrypi` and `nx` .

**`-musl`**

Enables musl libc compatibility. (Linux NG only)

**`-nostrictupgrade`**

Don't upgrade strict method void return types, if required. (NG only)

If a Strict sub type overrides the method of a SuperStrict type and the return type is void, don't

By default, the output file is placed into the same directory as the root source file.

**`-q`**

Quiet build.

**`-quick`**

Quick build.

Does not scan modules for changes. May result in quicker build times on some systems.

The default behaviour is to scan and build all requirements for the application, including modules.

**`-r`**

Builds a release version.

**`-standalone`**

Generate but do not compile into binary form.

Useful for creating ready-to-build source for a different platform/architecture.

**`-static`**

Statically link binary. (Linux NG only)

**`-t <app type>`**

Specifies the application type. (makeapp only)

Should be either `console` or `gui` .

The default is `console` .

**`-v`**

Verbose (noisy) build.

`-x`

Executes built application. (makeapp only)

`-k`

When using the `cleanmods` this param tells `bmk` to remove other files in the module folder too. Kept are: "i","a","txt","htm" and "html" files and the "doc" folder. Use with caution!

`-f <framework>`

Defines to use a specific module as framework instead of importing all `brl.mod` and `pub.mod` modules.

`-nomanifest`

Do not generate a manifest file for the built application (Win32 only).

`-single`

Disabled multi threaded processing in a `bmk` built with thread support. So tasks are processed in sequence rather than parallel.

`-nodef`

Defines to not generate .def files useable by created DLLs/shared libraries.

`-nohead`

Defines to not generate header files useable by created DLLs/shared libraries.

`-override`

Sets requirement for overriding methods and functions to append `override` to their definitions.

`-overerr`

Defines missing `override` keywords in overridden methods and functions to be handled as error instead of warning.

Compress the created binary with UPX (only if UPX binary is present in the /bin directory).

← MAXIDE

BLITZMAX COMPILER (BCC) →