BlitzMax Compiler (bcc)

EDIT

`bcc` is used to translate BlitzMax source code into C code. The name "bcc" came from Blitz Code Compiler but since BlitzMax NG `bcc` became more of a transpiler.

Command line syntax

```
bcc [options]
```

Copy

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

Options

`-q`

Tells `bcc` to be quiet. For now it does not change something in `bcc` output itself but if passed back to `bmk` this will have an effect to [`bmk`].

`-v`

Sets `bcc` into a verbose mode. This outputs more text information than during normal operation.

This can be useful if you want to see where stuff hangs, takes a while or just to see what happesn during compilation.

`-r`

Create a release mode transpile of the BlitzMax source code.

`-s`

Disable "strict upgrade". This then no longer upgrades strict subclass method/function return types to match the superstrict superclass. Without this parameter (so "default") `bcc` does auto-upgrade. Set flag if you want to throw an error - because of mismatch. (Example: strict is Int, superstrict is Void).

`-g <architecture>`

Select for what architecture to create code. Valid architectures (for now) are: `x86` . `x64` , `pcc` , `arm` , `arm64` , `armeabi` , `armeabiv7a` , and `arm64v8a` .

`-m <modulename>`

Create code for the given `modulename` .

`-o <output file>`

Specifies the output file. Must be full path to the outputfile (excluding final extension - there will be a .h, .c and .i generated).

`-p <platform>`

Select platform to create code for. Valid platforms (for now) are: Windows OS: `win32` Mac OS X: `macos` Linux: `linux` Android: `android` RaspberryPi: `raspberrypi` Web / Javascript: `emscripten`

`-t <apptype>`

Define type of the application. Should be either `console` or `gui` .

The default is `console` .

`-f <framework>`

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

`-d`

-w

Generate warnings (and accept) instead of errors for calling methods with arguments that need to be cast down. May cause issues using overloaded methods.

-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.

-makelib

Tells `bcc` that the generated code is used in a DLL/shared library.

-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.

-ud <user defined conditionals>

Add user defined conditionals (comma separated) then usable via `?myconditional` .

← BLITZMAX MAKE (BMK)