LOCAL\_EXPORT\_CPPFLAGS

Same as LOCAL\_EXPORT\_CFLAGS, but for C++ flags only.

LOCAL\_EXPORT\_C\_INCLUDES

Same as LOCAL\_EXPORT\_CFLAGS, but for C include paths.

This can be useful if 'bar.c' wants to include headers that are provided by module 'foo'.

LOCAL\_EXPORT\_LDLIBS

Same as LOCAL\_EXPORT\_CFLAGS, but for linker flags. Note that the imported linker flags will be appended to your module's LOCAL\_LDLIBS

though, due to the way Unix linkers work.

This is typically useful when module 'foo' is a static library and has code that depends on a system library. LOCAL\_EXPORT\_LDLIBS can then be

used to export the dependency. For example:

include $(CLEAR\_VARS)

LOCAL\_MODULE := foo

LOCAL\_SRC\_FILES := foo/foo.c

LOCAL\_EXPORT\_LDLIBS := -llog

include $(BUILD\_STATIC\_LIBRARY)

include $(CLEAR\_VARS)

LOCAL\_MODULE := bar

LOCAL\_SRC\_FILES := bar.c

LOCAL\_STATIC\_LIBRARIES := foo

include $(BUILD\_SHARED\_LIBRARY)

There, libbar.so will be built with a -llog at the end of the linker command to indicate that it depends on the system logging library,

because it depends on 'foo'.

LOCAL\_SHORT\_COMMANDS

Set this variable to 'true' when your module has a very high number of sources and/or dependent static or shared libraries. This forces the

build system to use an intermediate list file, and use it with the library archiver or static linker with the @$(listfile) syntax.

This can be useful on Windows, where the command-line only accepts a maximum of 8191 characters, which can be too small for complex

projects.

This also impacts the compilation of individual source files, placing nearly all compiler flags inside list files too.

Note that any other value than 'true' will revert to the default behaviour. You can also define APP\_SHORT\_COMMANDS in your

Application.mk to force this behaviour for all modules in your project.

NOTE: We do not recommend enabling this feature by default, since it makes the build slower.

LOCAL\_FILTER\_ASM

Define this variable to a shell command that will be used to filter the assembly files from, or generated from, your LOCAL\_SRC\_FILES.

When it is defined, the following happens:

- Any C or C++ source file is generated into a temporary assembly file (instead of being compiled into an object file).

- Any temporary assembly file, and any assembly file listed in LOCAL\_SRC\_FILES is sent through the LOCAL\_FILTER\_ASM command

to generate \_another\_ temporary assembly file.

- These filtered assembly files are compiled into object file.

In other words, If you have:

LOCAL\_SRC\_FILES := foo.c bar.S

LOCAL\_FILTER\_ASM := myasmfilter

foo.c --1--> $OBJS\_DIR/foo.S.original --2--> $OBJS\_DIR/foo.S --3--> $OBJS\_DIR/foo.o

bar.S --2--> $OBJS\_DIR/bar.S --3--> $OBJS\_DIR/bar.o

Were "1" corresponds to the compiler, "2" to the filter, and "3" to the assembler. The filter must be a standalone shell command that takes the

name of the input file as its first argument, and the name of the output file as the second one, as in:

myasmfilter $OBJS\_DIR/foo.S.original $OBJS\_DIR/foo.S

myasmfilter bar.S $OBJS\_DIR/bar.S

NDK\_TOOLCHAIN\_VERSION

Define this variable to either 4.4.3 or 4.6 to select version of GCC compiler. 4.6 is the default

The Android Build Cookbook offers code snippets to help you quickly implement some common build tasks. For additional instruction, please see the other build documents in this section.