LOCAL\_ARM\_NEON

Defining this variable to 'true' allows the use of ARM Advanced SIMD (a.k.a. NEON) GCC intrinsics in your C and C++ sources, as well as

NEON instructions in Assembly files.

You should only define it when targeting the 'armeabi-v7a' ABI that corresponds to the ARMv7 instruction set. Note that not all ARMv7

based CPUs support the NEON instruction set extensions and that you should perform runtime detection to be able to use this code at runtime

safely. To learn more about this, please read the documentation at docs/CPU-ARM-NEON.html and docs/CPU-FEATURES.html.

Alternatively, you can also specify that only specific source files may be compiled with NEON support by using the '.neon' suffix, as in:

LOCAL\_SRC\_FILES = foo.c.neon bar.c zoo.c.arm.neon

In this example, 'foo.c' will be compiled in thumb+neon mode, 'bar.c' will be compiled in 'thumb' mode, and 'zoo.c' will be

compiled in 'arm+neon' mode.

Note that the '.neon' suffix must appear after the '.arm' suffix if you use both (i.e. foo.c.arm.neon works, but not foo.c.neon.arm !)

LOCAL\_DISABLE\_NO\_EXECUTE

Android NDK r4 added support for the "NX bit" security feature.

It is enabled by default, but you can disable it if you \*really\* need to by setting this variable to 'true'.

NOTE: This feature does not modify the ABI and is only enabled on kernels targeting ARMv6+ CPU devices. Machine code generated

with this feature enabled will run unmodified on devices running earlier CPU architectures.

For more information, see:

http://en.wikipedia.org/wiki/NX\_bit

http://www.gentoo.org/proj/en/hardened/gnu-stack.xml

LOCAL\_DISABLE\_RELRO

By default, NDK compiled code is built with read-only relocations and GOT protection. This instructs the runtime linker to mark

certain regions of memory as being read-only after relocation, making certain security exploits (such as GOT overwrites) harder

to perform.

It is enabled by default, but you can disable it if you \*really\* need to by setting this variable to 'true'.

NOTE: These protections are only effective on newer Android devices ("Jelly Bean" and beyond). The code will still run on older

versions (albeit without memory protections).

For more information, see:

http://isisblogs.poly.edu/2011/06/01/relro-relocation-read-only/

http://www.akkadia.org/drepper/nonselsec.pdf (section 6)

LOCAL\_EXPORT\_CFLAGS

Define this variable to record a set of C/C++ compiler flags that will be added to the LOCAL\_CFLAGS definition of any other module that uses

this one with LOCAL\_STATIC\_LIBRARIES or LOCAL\_SHARED\_LIBRARIES.

For example, consider the module 'foo' with the following definition:

include $(CLEAR\_VARS)

LOCAL\_MODULE := foo

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

LOCAL\_EXPORT\_CFLAGS := -DFOO=1

include $(BUILD\_STATIC\_LIBRARY)

And another module, named 'bar' that depends on it as:

include $(CLEAR\_VARS)

LOCAL\_MODULE := bar

LOCAL\_SRC\_FILES := bar.c

LOCAL\_CFLAGS := -DBAR=2

LOCAL\_STATIC\_LIBRARIES := foo

include $(BUILD\_SHARED\_LIBRARY)

Then, the flags '-DFOO=1 -DBAR=2' will be passed to the compiler when building bar.c

Exported flags are prepended to your module's LOCAL\_CFLAGS so you can easily override them. They are also transitive: if 'zoo' depends on

'bar' which depends on 'foo', then 'zoo' will also inherit all flags exported by 'foo'.

Finally, exported flags are \*not\* used when building the module that exports them. In the above example, -DFOO=1 would not be passed to the

compiler when building foo/foo.c.