GYP->GN Conversion Cookbook

Contents

Targets

GYP	*GN*
`'type': 'static_library', 'name': 'foo',`	`static_library("foo") {` or `source_set("foo") {`
`'type': 'shared_library', 'name': 'foo',`	`shared_library("foo") {`
`'type': '<(component)', 'name': 'foo',`	`component("foo") {`
`'type': 'executable', 'name': 'foo',`	`executable("foo") {`
`'type': '<(gtest_target_type)', 'name': 'foo',`	`test("foo") {`
`'type': 'none', 'name': 'foo',`	`group("foo") {`

Note on static libraries

A source_set is basically a transparent static_library. The source files are compiled with the given options but not linked into anything. Targets that depend on a source set get the source set's object files linked into it. This saves the overhead of creating a static library on disk, avoids weird linker issues when a static library has no source files, and you can link source sets into shared libraries and have symbols exported from the shared library.

The last issue is a cause of a lot of headaches in the GYP build. If you annotate a symbol as exported (i.e. `BASE_EXPORT`) then you can't have it in a file that goes into a static library because the function might be stripped out if it's not called from within the static library. This prevents composing components of static libraries and exporting their symbols. A source set avoids this issue and `EXPORT` has the desired meaning of "export from the component this gets linked into" with no surprising dead code stripping behavior.

A disadvantage of source sets is that if an object file is completely unused, it will still be linked into the result, which is not the case for static libraries. A few places in the build depend on this behavior (deliberately or accidentally). In general, small libraries that we expect to be entirely used, test helpers, etc. can be source sets. There is slightly less risk of subtle issues if you keep static libraries static libraries, however.

Actions

GYP

```
{
  'action_name': 'foo',
  'inputs': [ 'bar.py' ],
  'outputs': [ '<(SHARED_INTERMEDIATE_DIR)/bar.out' ],
  'action': ['python', 'bar.py', '--la_dee_da' ],
},
```

Unlike GYP, where an action is a part of a target, GN actions are separate targets that you then depend on via deps from other targets:

```
action("foo") {
  script = "bar.py"
  outputs = [ "$target_gen_dir/bar.out" ]
  args = [ "--la_dee_da" ]
}

executable("foo.exe") {
  ...
  deps = [ ":foo" ]  # Depend on the action to make sure it runs.
}
```

Rules in GYP become `action_foreach` in GN which work like actions but iterate over a set of sources.

Copies

GYP

```
'copies': [
  {
    'destination': '<(PRODUCT_DIR)/',
    'files': [
      '../build/win/dbghelp_xp/dbghelp.dll',
    ],
  },
],
```

Unlike GYP, where copies are part of a target, GN copies are separate targets that you then depend on via deps from other targets:

```
copy("bar") {
  sources = [ "../path/to/secret.dll" ]
  outputs = [ "$root_out_dir/{{source_file_part}}" ]
}

component("base") {
  ...
  deps = [ "bar" }  # Depend on the copy to make sure it runs.
}
```

Platform checking

GYP	*GN*
`'conditions': [['OS=="win"', {`	`if (is_win) {`
`'conditions': [['OS=="linux"', {`	`if (is_linux) {`
`'conditions': [['OS=="android"', {`	`if (is_android) {`
`'conditions': [['OS=="mac"', {`	`if (is_mac) {`
`'conditions': [['OS=="ios"', {`	`if (is_ios) {`
`'conditions': [['chromeos==1', {`	`if (is_chromeos) {`

Typical sources and deps modifications

GYP

```
'sources': [
  'a.cc',
  'b.cc',
  'c.cc',
],
```

```
'dependencies': [
  '<(DEPTH)/base/base.gyp:foo',
],
'conditions': [
  ['OS=="win"': {
    'sources!': [
       'a.cc',
    ],
    'sources': [
      'foo.cc',
    ],
    'dependencies': [
      '<(DEPTH)/base/base.gyp:bar',
    ],
  }, {
    'sources/': [
      ['exclude', '^b\\.cc$'],
    ],
  }],
],
```

GN

```
sources = [
  "c.cc",
]
deps = [
  "//base:foo",
]

if (is_win) {
  sources += [
    "b.cc",
    "foo.cc',
  ]
  deps += [ "//base:bar" ]
} else {
  sources += [ "a.cc" ]
}
```

Note that in GN we prefer to only add files when needed, and don't add all of them at first only to remove them later like in gyp.

Variable mappings

Build configuration

Build configuration and feature flags are usually global in GYP. In GN we try to limit global variables and instead put variables used by only some files into `.gni` files. These files are then imported into your buildfile by specifying at the top:

```
import("//build/config/features.gni")

# ... now you can use the variables declared in features.gni.
if (is_tsan) {
  # ...
}
if (cld_version == 2) {
  # ...
}
```

Other flags only apply to one `BUILD.gn` file and those flags are declared directly in that file (so other files can't use them). These places are noted in the table below.

GYP	*GN*	*GN import*
`arm_float_abi`	`arm_float_abi`	`//build/config/arm.gni`
`arm_neon` (0/1)	`arm_use_neon` (true/false)	`//build/config/arm.gni`
`arm_neon_optional` (0/1)	`arm_optionally_use_neon` (true/false)	`//build/config/arm.gni`
`arm_version`	`arm_version`	`//build/config/arm.gni`
`asan` (0/1)	`is_asan` (true/false)	`//build/config/sanitizers/sanitizers.gni`

branding ("Chromium"/"Chrome")	is_chrome_branded (true/false)	//build/config/chrome_build.gni
build_for_tool=="drmemory"	enable_iterator_debugging=false	(internal to //build/config/BUILD.gn)
build_for_tool=="tsan"	enable_iterator_debugging=false	(internal to //build/config/BUILD.gn)
buildtype ("Official"/"Dev")	is_official_build (true/false)	//build/config/chrome_build.gni
chrome_multiple_dll (0/1)	is_multi_dll_chrome (true/false)	//build/config/chrome_build.gni
clang (0/1)	is_clang (true/false)	(global)
clang_use_chrome_plugins (0/1)	clang_use_chrome_plugins (true/false)	(internal to //build/config/clang/BUILD.gn)
component ("shared_library"/"static_library")	is_component_build (true/false)	(global)
desktop_linux (0/1)	is_desktop_linux (true/false)	(global)
disable_glibcxx_debug (0/1)	enable_iterator_debugging (true/false)	(internal to //build/config/BUILD.gn)
fastbuild (0/1/2)	symbol_level (2/1/0 — values inverted)	//build/config/compiler/compiler.gni
gomadir	goma_dir	//build/toolchain/goma.gni
ios_deployment_target (string)	ios_deployment_target	//build/config/ios/ios_sdk.gni
GYP_MSVS_OVERRIDE_PATH environment variable	visual_studio_path	//build/config/win/visual_studio_version.gni
GYP_MSVS_VERSION environment variable	(none)	
ios_sdk_path	ios_sdk_path and use_ios_simulator	//build/config/ios/ios_sdk.gni
lsan (0/1)	is_lsan (true/false)	//build/config/sanitizers/sanitizers.gni
mac_sdk_min	mac_sdk_min	//build/config/mac/mac_sdk.gni
mac_sdk_path	mac_sdk_path	//build/config/mac/mac_sdk.gni
mac_sdk	mac_sdk_version	//build/config/mac/mac_sdk.gni
msan (0/1)	is_msan (true/false)	//build/config/sanitizers/sanitizers.gni
SDKROOT (Mac)	sysroot	//build/config/sysroot.gni
sysroot	sysroot	//build/config/sysroot.gni
target_arch ("ia32"/"x64"/"arm"/"mipsel")	target_cpu ("x86"/"x64"/"arm"/"mipsel")	(global)
toolkit_views (0/1)	toolkit_views	//build/config/ui.gni
tsan (0/1)	is_tsan (true/false)	//build/config/sanitizers/sanitizers.gni
windows_sdk_path	windows_sdk_path	(internal to //build/config/win/BUILD.gn)

Feature flags

GYP	*GN*	*GN import*
cld_version (number)	cld_version (number)	//build/config/features.gni
configuration_policy (0/1)	enable_configuration_policy (true/false)	//build/config/features.gni
debug_devtools (0/1)	debug_devtools (true/false)	//build/config/features.gni
disable_ftp_support (0/1)	disable_ftp_support (true/false)	//build/config/features.gni
disable_nacl (0/1)	enable_nacl (true/false)	//build/config/features.gni
enable_app_list (0/1)	enable_app_list (true/false)	//build/config/features.gni
enable_autofill_dialog (0/1)	enable_autofill_dialog (true/false)	//build/config/features.gni
enable_background (0/1)	enable_background (true/false)	//build/config/features.gni
enable_captive_portal_detection (0/1)	enable_captive_portal_detection (true/false)	//build/config/features.gni
enable_chromevox_next (0/1)	enable_chromevox_next (true/false)	//build/config/features.gni
enable_extensions (0/1)	enable_extensions (true/false)	//build/config/features.gni
enable_google_now (0/1)	enable_google_now (true/false)	//build/config/features.gni
enable_hidpi (0/1)	enable_hidpi (true/false)	//ui/base/ui_features.gni
enable_managed_users (0/1)	enable_managed_users (true/false)	//build/config/features.gni
enable_mdns (0/1)	enable_mdns (true/false)	//build/config/features.gni
enable_one_click_signin (0/1)	enable_one_click_signin	//chrome/common/features.gni

	(true/false)			
enable_pepper_cdms (0/1)	enable_pepper_cdms (true/false)	//build/config/features.gni		
enable_plugins (0/1)	enable_plugins (true/false)	//build/config/features.gni		
enable_plugin_installation (0/1)	enable_plugin_installation (true/false)	//build/config/features.gni		
enable_basic_printing (0/1)	enable_basic_printing (true/false)	//build/config/features.gni		
enable_print_preview (0/1)	enable_print_preview (true/false)	//build/config/features.gni		
enable_rlz (0/1)	enable_rlz (true/false)	//build/config/features.gni		
enable_service_discovery (0/1)	enable_service_discovery (true/false)	//build/config/features.gni		
enable_spellcheck (0/1)	enable_spellcheck (true/false)	//build/config/features.gni		
enable_session_service (0/1)	enable_session_service (true/false)	//build/config/features.gni		
enable_settings_app (0/1)	enable_settings_app (true/false)	//build/config/features.gni		
enable_task_manager (0/1)	enable_task_manager (true/false)	//build/config/features.gni		
enable_themes (0/1)	enable_themes (true/false)	//build/config/features.gni		
enable_webrtc (0/1)	enable_webrtc (true/false)	//build/config/features.gni		
image_loader_extension (0/1)	enable_image_loader_extension (true/false)	//build/config/features.gni		
input_speech (0/1)	enable_speech_input (true/false)	//build/config/features.gni		
notifications (0/1)	enable_notifications (true/false)	//build/config/features.gni		
ozone_platform_dri (0/1)	ozone_platform_dri (true/false)	//build/config/ui.gni		
remoting (0/1)	enable_remoting (true/false)	//build/config/features.gni		
safe_browsing (0/1/2)	safe_browsing_mode (0/1/2)	//build/config/features.gni		
use_allocator ('none'	'tcmalloc')	use_allocator ("none"	"tcmalloc")	(See "Allocator" below)
ui_compositor_image_transport (0/1)	ui_compositor_image_transport (true/false)	//build/config/ui.gni		
use_ash (0/1)	use_ash (true/false)	//build/config/ui.gni		
use_athena (0/1)	use_athena (true/false)	//build/config/ui.gni		
use_aura (0/1)	use_aura (true/false)	//build/config/ui.gni		
use_brlapi (0/1)	use_brlapi (true/false)	//build/config/features.gni		
use_cairo (0/1)	use_cairo (true/false)	//build/config/ui.gni		
use_clipboard_aurax11 (0/1)	use_aura && use_x11			
use_cups (0/1)	use_cups (true/false)	//build/config/features.gni		
use_dbus (0/1)	use_dbus (true/false)	//build/config/features.gni		
use_gconf (0/1)	use_gconf (true/false)	//build/config/features.gni		
use_glib (0/1)	is_linux (true/false)	(global)		
use_gnome_keyring (0/1)	is_desktop_linux (true/false)			
use_goma (0/1)	use_goma (true/false)	//build/toolchain/goma.gni		
use_nss_certs (0/1)	use_nss_certs (true/false)	//build/config/crypto.gni (Many of these conditions can be deleted, see the "SSL" notes on targets below.)		
use_pango (0/1)	use_pango (true/false)	//build/config/ui.gni		
use_ozone (0/1)	use_ozone (true/false)	//build/config/ui.gni		
use_seccomp_bpf (0/1)	use_seccomp_bpf (true/false)	//build/config/features.gni		
use_udev (0/1)	use_udev (true/false)	//build/config/features.gni		
use_x11 (0/1)	use_x11 (true/false)	//build/config/ui.gni		
use_xi2_mt (0/1)	use_xi2_mt (true/false)	//build/config/ui.gni		
win_use_allocator_shim (0/1)		(See "Allocator" below)		

Common target conversion

Some targets that lots of projects depend on and how the GN ones correspond to GYP ones. (This is for commonly-depended-on or weird targets only, don't add stuff here just because you converted it.)

GYP	*GN*	*Notes* **(see below)**
base/base.gyp:base	//base	
base/base.gyp:base_i18n	//base:i18n	
base/base.gyp:run_all_unittests	//base/test:run_all_unittests	
base/base.gyp:test_support_base	//base/test:test_support	
base/third_party/dynamic_annotations/dynamic_annotations.gyp:dynamic_annotations	//base/third_party/dynamic_annotations	
build/linux/system.gyp:* (except nss)	//build/config/linux:*	Linux system targets
build/linux/system.gyp:nss	//crypto:platform	SSL
net/third_party/nss/ssl.gyp:libssl	//crypto:platform	SSL
skia/skia.gyp:skia	//skia	
testing/gmock.gyp:gmock	//testing/gmock	Secondary tree
testing/gtest.gyp:gtest	//testing/gtest	Secondary treeo
third_party/icu/icu.gyp:icui18n	//third_party/icu	Secondary tree, ICU
third_party/icu/icu.gyp:icuuc	//third_party/icu	Secondary tree, ICU
url/url.gyp:url_lib	//url	

Notes:

- *ICU:* GN has separate `//third_party/icu:icuuc` and `//third_party/icu:icui18n` targets just like GYP. You can use these if you only need one of them. Most targets want both, so GN made a meta target that's just `//third_party/icu` which you can use that redirects to both "uc" and "i18n".
- *Linux system targets:* Generally the names in GN patch the GYP names for the Linux system-related stuff. However, most of them are configs instead of actual targets (in GYP they're all targets). For example, since "x11" is just a list of libraries and include directories, and includes no sources it's a config that just adds this configuration to your target. To use a config, do `configs += [ "//build/config/linux:x11" ]`
- *Secondary tree:* Some projects are DEPSed in and we want it to look like a BUILD.gn file is in that directory without checking it in to the upstream repo. The directory `build/secondary` mirrors the main tree and is checked for BUILD.gn files if an expected file in the main tree wasn't found.
- *SSL:* In GYP there are lots of conditions around NSS vs. OpenSSL and different platforms that determine which of the different SSL libraries is linked. In GN, there is a meta-target `//crypto:platform` that will "do the right thing" according to the current build platform and flags. Generally its safe to replace any conditional reference to a SSL library with this one.

Visibility and header file issues

GN is much more strict about header file checking. You may encounter errors that your target doesn't depend on the target containing a certain header file. The most common example is including `base/macros.h` without having `//base` in your project's dependency list. The solution is to just add the missing dependency.

The dependency tree must be a DAG. Some components might share headers between a number of internal targets that makes adding the "proper" dependencies impossible. In this case, you can separate out a `source_set` type target containing just the header(s) in question, and make the targets that use that header depend on that source set to break the cycle.

Other stuff

Target conditions

`target_conditions` are like normal conditions but expanded in a different phase of GYP. You can generally just convert the conditions inside and not worry about the `conditions` / `target_conditions` difference.

xcode_settings

Some xcode settings are obvious:

```
  'xcode_settings': {'OTHER_LDFLAGS': ['-foo']},
```

Should just expand to:

```
  ldflags = [ "-foo" ]
```

Other flags are less obvious:

```
  'xcode_settings': { 'GCC_SYMBOLS_PRIVATE_EXTERN': 'NO', },
```

These all correspond to various flags that get passed to the compiler or linker. You can use your favorite search engine to see what it corresponds to, but many of them are not well documented. You can also search for the string in tools/gyp/pylib/gyp/xcode_emulation.py. GYP uses this file to decode the Xcode settings into command line flags for the ninja build.

wexit-time destructors

Replace

```
'enable_wexit_time_destructors': 1,
```

with

```
configs += [ "//build/config/compiler:wexit_time_destructors" ]
```

Chromium code

In GYP code is "non-Chromium" by default, and you opt into higher warning levels using:

```
'chromium_code': 1,
```

In GN, all code is Chromium code by default. If you're compiling a static library that needs more lax warnings, opt out of the Chromium-code settings with:

```
configs -= [ "//build/config/compiler:chromium_code" ]
configs += [ "//build/config/compiler:no_chromium_code" ]
```

-fvisibility

All symbols in the build have "hidden" visibility by default (this means that symbols aren't exported from shared libraries, a concept different than GN's target visibility). If you needed to export all symbols (for a third party library) by default in GYP you would do:

```
'xcode_settings': [
  'GCC_SYMBOLS_PRIVATE_EXTERN': 'NO',  # no -fvisibility=hidden
],
'cflags!': [
  '-fvisibility=hidden',
],
```

In GN the equivalent is:

```
if (!is_win) {
  configs -= [ "//build/config/gcc:symbol_visibility_hidden" ]
}
```

Dependent settings

In GYP you'll see stuff like this, especially in third-party code.

```
'direct_dependent_settings': {
  'include_dirs': [
    '.',  # This directory.
    '../..',  # Root "src" path.
  ],
  'defines': [
    'FOO',
  ],
},
```

Note that many of the includes are trying to add the root "src" directory to the include path. This is always present in GN so you can remove these.

GYP also requires you to duplicate these settings, once for the target itself, and once for the direct/all dependent settings. In GN, public/all dependent configs also apply to the current target so you only need to specify it once.

In GN, put the settings in a config (declared above your target), and then reference that as a public config in your target:

```
config("foo_config") {
  include_dirs = [ "." ]
  defines = [ "FOO" ]
}

component("foo") {
  ...
  public_configs = [ ":foo_config" ]
}
```

Targets that depend on `foo` will now get `foo_config` applied.

GYP would say `export_dependent_settings` to forward `direct_dependent_settings` up the dependency chain. In GN, put the dependency in the `public_deps` section and this will happen automatically.

MSVS disabled warnings

In GYP you'll see for third-party code:

```
'msvs_disabled_warnings': [ 4018, 4244, 4267, ],
```

At least half of the warnings in these blocks are already disabled globally (we added more global ones later). From the command line, do:

```
$ cd src/build/config
$ git grep 4018
compiler/BUILD.gn:      "/wd4018",  # Comparing signed and unsigned values.
```

tells us that warning 4018 is already disabled globally from the `default_warning_flags` variable in `//build/config/compiler`, and the same for 4244. So ignore these.

Always comment what the warning is. Use your favorite search engine and type "vc warning 4267" to look it up. You'll end up with:

```
if (is_win) {
  cflags += [
    "/wd4267",  # Conversion from size_t to 'type'.
  ]
}
```

(Use `=` instead of `+=` is you haven't already defined a `cflags` variable.)

Mac frameworks

GN knows to convert `.framework` files in the `libs` list to the right thing on Mac. You don't need to specify the directories either. So convert this:

```
'link_settings': {
  'libraries': [
    '$(SDKROOT)/System/Library/Frameworks/Accelerate.framework',
  ],
},
```

to this:

```
libs = [ "Accelerate.framework" ]
```

hard_dependency

GYP code sometimes sets

```
  'hard_dependency': 1,
```

to indicate that the current target must be build before its dependents. GN can deduce this internally, so you can ignore this directive.

Allocator

GYP has `win_use_allocator_shim` and `use_allocator`. In GN, these are merged into `use_allocator` which is defined in `//build/config/allocator.gni`. *However* you should almost never need to use this flag. The `//base/allocator` target will change depending on the current allocator flag, so you can unconditionally depend on this target to pick up the current build defaults.

This:

```
  ['use_allocator!='none'', {
    'dependencies': [ '../base/allocator/allocator.gyp:allocator' ]
  }],
  ['win_use_allocator_shim==1', {
    'dependencies': [ '<(allocator_target)' ],
  }],
```

Becomes:

```
  deps = [ "//base/allocator" ]
```

As in GYP, the allocator should only be depended on by executables (and tests). Libraries should not set the allocator.

optimize: max

In Gyp:

```
'optimize': 'max',
```

only affects Windows and will optimize for speed rather than size. To get the same behavior in GN, do:

```
if (!is_debug && is_win) {
  configs -= [ "//build/config/compiler:default_optimization" ]
  configs += [ "//build/config/compiler:optimize_max" ]
}
```

The `is_win` check is needed because the `optimize_max` config also affects Posix systems. Some components might additionally specify `-O2` on Posix further optimize, in which case you can remove the `is_win` check.

Protobufs

```
import("//third_party/protobuf/proto_library.gni")

proto_library("myproto") {
  sources = [ "foo.proto" ]
}
```

See the `third_party/protobuf/proto_library.gni` file for full documentation and extra flags.

Java stuff

JNI generator in GYP:

```
{
  'target_name': 'foo_headers',
  'type': 'none',
  'sources': [ <java files> ]
  'variables': { 'jni_gen_package': 'foobar' }
  'includes': [ 'build/jni_generator.gypi' ]
}
```

JNI generator in GN:

```
# At top of file:
if (is_android) {
  import("//build/config/android/rules.gni")
}

# Later:
if (is_android) {
  generate_jni("foo_headers") {
    sources = [ <java files> ]
    jni_package = "foobar"
  }
}
```

Grit

```
import("//tools/grit/grit_rule.gni")

grit("resources") {
  source = "my_resources.grd"
}
```

See `src/build/secondary/tools/grit/grit_rule.gni` for more documentation.

Mojo

```
import("//mojo/public/tools/bindings/mojom.gni")

mojom("mojo_bindings") {
  sources = [
    "foo.mojom",
  ]
}
```

Powered by Gitiles source log blame