GYP

# vs. CMake

GYP was originally created to generate native IDE project files (Visual Studio, Xcode) for building Chromium.

The functionality of GYP is very similar to the CMake build tool. Bradley Nelson wrote up the following description of why the team created GYP instead of using CMake. The text below is copied from http://www.mail-archive.com/webkit-dev@lists.webkit.org/msg11029.html

```
Re: [webkit-dev] CMake as a build system?
Bradley Nelson
Mon, 19 Apr 2010 22:38:30 -0700

Here's the innards of an email with a laundry list of stuff I came up with a
while back on the gyp-developers list in response to Mike Craddick regarding
what motivated gyp's development, since we were aware of cmake at the time
(we'd even started a speculative port):


I did an exploratory port of portions of Chromium to cmake (I think I got as
far as net, base, sandbox, and part of webkit).
There were a number of motivations, not all of which would apply to other
projects. Also, some of the design of gyp was informed by experience at
Google with large projects built wholly from source, leading to features
absent from cmake, but not strictly required for Chromium.

1. Ability to incrementally transition on Windows. It took us about 6 months
to switch fully to gyp. Previous attempts to move to scons had taken a long
time and failed, due to the requirement to transition while in flight. For a
substantial period of time, we had a hybrid of checked in vcproj and gyp gene
vcproj. To this day we still have a good number of GUIDs pinned in the gyp fi
because different parts of our release pipeline have leftover assumptions
regarding manipulating the raw sln/vcprojs. This transition occurred from
the bottom up, largely because modules like base were easier to convert, and
had a lower churn rate. During early stages of the transition, the majority
of the team wasn't even aware they were using gyp, as it integrated into
their existing workflow, and only affected modules that had been converted.
```

2. Generation of a more 'normal' vcproj file. Gyp attempts, particularly on Windows, to generate vcprojs which resemble hand generated projects. It doesn't generate any Makefile type projects, but instead produces msvs Custom Build Steps and Custom Build Rules. This makes the resulting projects easier to understand from the IDE and avoids parts of the IDE that simply don't function correctly if you use Makefile projects. Our early hope with gyp was to support the least common denominator of features present in each of the platform specific project file formats, rather than falling back on generated Makefiles/shell scripts to emulate some common abstraction. CMake by comparison makes a good faith attempt to use native project features, but falls back on generated scripts in order to preserve the same semantics on each platforms.

3. Abstraction on the level of project settings, rather than command line flags. In gyp's syntax you can add nearly any option present in a hand generated xcode/vcproj file. This allows you to use abstractions built into the IDEs rather than reverse engineering them possibly incorrectly for things like: manifest generation, precompiled headers, bundle generation. When somebody wants to use a particular menu option from msvs, I'm able to do a web search on the name of the setting from the IDE and provide them with a gyp stanza that does the equivalent. In many cases, not all project file constructs correspond to command line flags.

4. Strong notion of module public/private interface. Gyp allows targets to publish a set of direct_dependent_settings, specifying things like include_dirs, defines, platforms specific settings, etc. This means that when module A depends on module B, it automatically acquires the right build settings without module A being filled with assumptions/knowledge of exactly how module B is built. Additionally, all of the transitive dependencies of module B are pulled in. This avoids their being a single top level view of the project, rather each gyp file expresses knowledge about its immediate neighbors. This keep local knowledge local. CMake effectively has a large shared global namespace.

5. Cross platform generation. CMake is not able to generate all project files on all platforms. For example xcode projects cannot be generated from windows (cmake uses mac specific libraries to do project generation). This means that for instance generating a tarball containing pregenerated projects for all platforms is hard with Cmake (requires distribution to several machine types).

6. Gyp has rudimentary cross compile support. Currently we've added enough functionality to gyp to support x86 -> arm cross compiles. Last I checked this functionality wasn't present in cmake. (This occurred later).

That being said there are a number of drawbacks currently to gyp:

1. Because platform specific settings are expressed at the project file

```
level (rather than the command line level). Settings which might otherwise
be shared in common between platforms (flags to gcc on mac/linux), end up
being repeated twice. Though in fairness there is actually less sharing here
than you'd think. include_dirs and defines actually represent 90% of what
can be typically shared.

2. CMake may be more mature, having been applied to a broader range of
projects. There a number of 'tool modules' for cmake, which are shared in a
common community.

3. gyp currently makes some nasty assumptions about the availability of
chromium's hermetic copy of cygwin on windows. This causes you to either
have to special case a number of rules, or swallow this copy of cygwin as a
build time dependency.

4. CMake includes a fairly readable imperative language. Currently Gyp has a
somewhat poorly specified declarative language (variable expansion happens
in sometimes weird and counter-intuitive ways). In fairness though, gyp assume
that external python scripts can be used as an escape hatch. Also gyp avoids
a lot of the things you'd need imperative code for, by having a nice target
settings publication mechanism.

5. (Feature/drawback depending on personal preference). Gyp's syntax is
DEEPLY nested. It suffers from all of Lisp's advantages and drawbacks.

-BradN
```

Powered by Gitiles