Wee8: Wasm C/C++ API in V8

With version 7.8, V8 introduces official[1] built-in support for the Wasm C/C++ API proposal.

***Warning**: Since this API is designed in a separate project, V8 can't make any promises about its stability. It is possible that you will have to update your embedding code if the API changes. V8 will aim to always support the latest version of the API.*
***2023 update**: V8-side work on this is currently unstaffed. We'll fix bugs, but aren't currently planning to work on improvements. Contrary to the "aim" stated above, the version of the API that V8 implements has fallen slightly behind upstream. Sorry!*
Comment access has been locked down due to vandalism. If you have any comments or questions, please send an email to jkummerow@chromium.org.

V8 provides a separate build target producing a library called libwee8. This library supports *only* WebAssembly, no JavaScript[2]. Here's how to use it:

Step 0: Get a V8 checkout (TL;DR: put depot_tools in $PATH and `fetch v8 && cd v8`).

Step 1: Create a build output directory, e.g. `out/wee8`, by running:

```
gn args out/wee8
```

Enter the build args you want, then save and quit the editor.
Recommended GN args are:

```
is_component_build = false  # shared library not yet supported
use_custom_libcxx = false  # use libstdc++, not libc++
v8_enable_fast_mksnapshot = true  # save some build time
v8_enable_i18n_support = false  # save some binary size
v8_use_external_startup_data = false  # monolithic bundle
```

For Debug builds, you will also want:

```
is_debug = true
symbol_level = 2
v8_optimized_debug = false
```

Whereas for Release builds, these make sense:

```
is_debug = false
symbol_level = 1  # or even 0
v8_enable_handle_zapping = false
```

There currently seems to be an issue where sometimes symbols aren't visible. If you encounter that, you can work around it by adding:

```
v8_expose_symbols = true
```

---

[1] Tip-of-tree builds have had experimental support since May 2019.
[2] Technically right now it still does support JavaScript, but we *will* break that in the future in order to reduce binary size, so please don't start building applications that rely on it!

But note that this is a temporary workaround; if you know how to repro this situation please let us know so we can develop a proper fix.

Step 2: Build libwee8.

```
autoninja -C out/wee8 wee8
```

Step 3: Write your embedding application (below: my.cc) and link it against libwee8. Example for a debug build with Clang and C++ (where $HEADERS is the directory that contains wasm.h and wasm.hh):

```
clang++ -std=c++11 -c my.cc -o my.o -O0 -ggdb -I $HEADERS
clang++ my.o -o my /path/to/v8/out/wee8/obj/libwee8.a -ldl -pthread
```

Release builds, GCC builds, and C builds work similarly. See V8's `tools/run-wasm-api-tests.py` for a variety of possible combinations.

Feedback, good or bad? Send email to jkummerow@chromium.org. I'd love to hear about your use cases, as well as any issues you encounter.

—

Appendix: Comments and discussions

Location of this documentation.
It has been suggested that this document could live on v8.dev/docs, to make it more discoverable and/or more official-looking.

Name of the library.
It has been suggested to name the built library "libwasm.a" (to match the wasm.h header) instead of libwee8.a. That would make it easier for projects to use it interchangeably with other implementations of the same API. OTOH a project-specific name might be less confusing. Further opinions on this would be appreciated.

Shared-library (.so / .dll) build.
On the to-do list.

About the `use_custom_libcxx = false` build arg.
Some people reported not needing it. It's required for linking libwee8.a with projects that use the regular glibc though. YMMV.