FFIgenpad

Mudit Somani

March 28, 2024

1 An Introduction

Name: Mudit Somani
GitHub: @thecomputerm
Email: mudit.somani00@gmail.com
Phone: +91 9920653004
University: BITS Pilani, Hyderabad Campus
Timezone: IST (UTC +5:30)
Primary Language: English

I am a second year undergrad pursuing a Computer Science degree at BITS Pilani Hyderabad, India. My programming expertise mainly lies with Dart, JavaScript, Go and C/C++.

I have actively participated in open-source projects for a reasonably long time and found great satisfaction in making contributions. I have been a part of Google Code-in 2019-20 where I ended up as a finalist with the Julia programming language. Additionally, I have hands-on experience in maintaining a Material UI component library for Svelte known as Svelte Materialify, was once one of the most popular UI libraries within the Svelte ecosystem. I currently maintain some college open-source projects and the official awesome list for svelte.

I have been enjoying using Dart, primary with Flutter, and I would love the opportunity to create something for the Dart ecosystem and community.

2 Sample Project

This GitHub Gist should demonstrate my understanding of the project and the necessary tools. I have also listed some additional experiments in the comments of the gist.

- Interop between C and Dart
- Filesystem in WASM

In addition to that, I have also contributed to fixing some issues.

- https://dart-review.googlesource.com/c/sdk/+/357801
- https://dart-review.googlesource.com/c/sdk/+/358080

I also made a online dart ast viewer that runs on WebAssembly (without a *backend*) which gives rudimentary information on the ast tree as generated by dart:analyzer. Here is the demo link and source code (private for now, please don't hesitate to email me in case something is wrong or is missing).

3 Problem Abstract

The web is a universal platform, and we see a lot of small utilities online that help in various tasks such as inspecting/debugging code or seeing the output of a specific function without the need to make a new project or use print statements. These can range from small utilities like formatting code without installing anything (prettier) and inspecting ASTs of some languages (ast explorer) to full nodejs runtimes (webcontainers) that fully run on the client in the browser.

I personally have used such tools for dart (like json to dart) and having the ability to use FFIgen on the web would be very useful for people working with native development in dart. However doing so is not trivial, as the technologies that would be used are still experimental, this brings me to my second point.

With the growing support for WASM and WasmGC, it is now possible for almost any language to be compiled down to performant WASM instead of using slow JS to make performance critical applications that can be distributed easily. I believe Dart should fully embrace this technology, however

the support for WebAssembly is still experimental. I believe this project proposal will also help to make WASM a better supported build target by identifying edge-cases and bugs that could happen during its development.

I have written more about what I aim to achieve in the below section.

4 Implementation Details

Here is a rough idea of what I imagine the steps needed to make FFIgenpad would look like.

1) Compile libclang to a WASM module.
2) Generate bindings for libclang in dart that can be compiled to WASM.
3) Override the libclang bindings used in ffigen to the one generated in step 2 and make sure to set the correct values for config for the WASM build (such as compiler-opts).
4) Create the web frontend with the ability to edit header files, edit config and display generated ffi bindings.
5) Integrate the web frontend.
6) Smooth out support for ffi generation for Objective-C in the app.
7) Style the app to make sure it is easy to use.

The steps appear straightforward, but given that WASM support is still experimental and the project will use bleeding-edge features such as the `@Native` annotation, I anticipate encountering bugs. I hope to address and resolve these issues and enhance WASM's support as a by-product of this project.

Here is what I imagine the user experience for this app will be:

- You can edit the config as yaml on the app.
- Will have two modes for ffi generation: C and Obj-C.
- Support for instant ffi generations using debounce.
- Ability to add header files by either writing them in textarea inputs or uploading them from your local system.

During this project I also hope to create blogs and tutorials on integrating WebAssembly with Dart. This will also encourage users to try out WASM, who don't do so due to the lack of documentation and help us prepare for when Dart's support for WASM moves beyond the experimental stage.

During this project, I will also try to implement other useful online tools that will help out the dart ecosystem, for example I hope to improve upon my Dart AST viewer as I think it will be very helpful for developers working with the dart AST (in projects like dart's analysis server) to quickly inspect the source code tree without any installations and avoiding the dart team incurring additional server costs (due to the app being fully client side).

5 Timeline

5.1 Community Bonding Period: May 1 - May 26

Please note that I have my final exams during this period due to which I have had to reduce the scope of the Goals. I will, however, be available on email, google groups, and GitHub for discussions.

Goals:

- Creating a more detailed outline of the project.
- Doing some groundwork on compiling libclang to WASM.
- Finalize the basic features and capabilities of the project.

5.2 Official Coding Period

As many of the technologies used in this project are still experimental, I am assuming that I will encounter issues relating to dart2wasm and js interop which I definitely aim to resolve (or at least help to resolve) while working on the project.

5.2.1 Week 1: May 27 - June 2

Create a WASM build of libclang that is able to be used on the web.

5.2.2 Week 2: June 3 - June 9

There is a good chance that the task from week 1 might extend to week 2 as this task is one of the more crucial and time consuming ones due to the lack of documentation.

5.2.3 Week 3: June 10 - June 16

Generate dart bindings for libclang with '@Native' annotations using the 'ffi-native' option so that we can use libclang our dart app.

5.2.4 Week 4: June 17 - June 23

Use the generated bindings to create a demo app as a proof of concept that libclang works properly on the web and also write some tutorials/blogs detailing interop between dart/wasm/js.

5.2.5 Week 5: June 24 - June 30

Buffer period as the tasks in week 1 and 2 or unexpected bugs might take more time than expected.

5.2.6 Week 6: July 1 - July 7

This week, I will finish documenting my code and submit my progress so far for the midterm evaluation.

5.2.7 Midterm Evaluations: July 8

Expected Results:

- WASM module for libclang that can be used on the web.
- Proof of concept that libclang can be used inside a dart app.
- Guides/blogs to better help other developers to quickly start using WASM in dart.

5.2.8 Week 7: July 8 - July 14

Override FFIgen's libclang bindings so that they can be used in the WASM build.

5.2.9 Week 8: July 15 - July 21

I expect the task in week 7 to extend to week 8 as well.

5.2.10 Week 9: July 22 - July 28

Work on integrating ffigen into the web app.

5.2.11 Week 10: July 29 - August 4

Iron out issues related to config options and ensure that even the obj-c bindings are being generated properly.

5.2.12 Week 11: August 5 - August 11

Possibly work on and explore to create similar tools like my dart AST analyzer and JNIgenpad which will also help developers. This work can atleast lay a foundation that other people (or me) can build up on in the future.

5.2.13 Week 12: August 12 - August 18

Design and implement a good UI for the web app to make it easy and intuitive to use.

5.2.14 Final Week: August 19 - August 26

Buffer period for any work left or any hard bugs.

5.3 Final Evaluation

Expected Results (in addition to midterm evaluation results):

- FFIgenpad is hosted online and functioning as expected.
- Possibly Dart AST Viewer is hosted online and functioning as expected.
- Tutorials/Guides on how to properly and efficiently use WASM as a build target for dart.
- Documentation of all the work done.