Automating ICU4X's Web Demo from Foreign Function Interface Definitions

Tyler Knowlton (ambiguousname@ambiguous.name)

Abstract

ICU4X's WASM bindings provide a powerful tool for designers and developers to create hassle-free localization for software. To encourage greater adoption, online playgrounds are always the first step in demonstrating versatility and flexibility. For these purposes, this proposal explores automating ICU4X's online WASM demo through pre-existing FFI definitions.

Background

ICU4X marks FFI definitions for Rust Diplomat, which allows use across nearly any conceivable language. This includes WebAssembly, which makes the creation of a web demonstration for ICU4X's capabilities remarkably easy: https://unicode-org.github.io/icu4x/wasm-demo/index.html.

Of course, a hand-written demo does not allow for easy scaling. The nice thing about an FFI definition, however, is the ability to see each possible function that could be called. From those definitions, we can generate examples for use.

Take for instance, `fixed-decimal.ts`. It makes use of several FFIs, the main two being: `ICU4XFixedDecimalFormatter.create_with_grouping_strategy`, which creates a formatter that allows us to call `format(ICU4XFixedDecimal)`. There is nothing that prevents an enterprising developer from opening the browser console and running these functions themselves. However, a UI does far more to visually demonstrate the possibilities in using such functions.

Creating UI from FFI definition

In the simplest version of an automatic demonstration, we would:

First, identify `ICU4XFixedDecimalFormatter.format` from the FFI definition. We know that this function has a visual component that would be great to showcase to an end user interested in testing ICU4X.

From there, we look at the function definition:

```rust
/// Formats a [`ICU4XFixedDecimal`] to a string.
#[diplomat::rust_link(
    icu::decimal::FixedDecimalFormatter::format, FnInStruct
)]
#[diplomat::rust_link(
    icu::decimal::FixedDecimalFormatter::format_to_string,
    FnInStruct,
    hidden
)]
#[diplomat::rust_link(
    icu::decimal::FormattedFixedDecimal, Struct, hidden)
]
#[diplomat::rust_link(
    icu::decimal::FormattedFixedDecimal::write_to,
    FnInStruct,
    hidden
)]
pub fn format(
    &self,
    value: &ICU4XFixedDecimal,
    write: &mut diplomat_runtime::DiplomatWriteable,
) -> Result<(), ICU4XError> {
    self.0.format(&value.0).write_to(write)?;
    Ok(())
}
```

Looking at the definition, `.format` requires a reference to `&self`, and a value `&ICU4XFixedDecimal`.

For simplicity's sake, let's say we find constructors for both. There are further chains of dependencies. Some should be pre-constructed, such as `ICU4XDataProivder`. We take note of every dependency and how it fits into user experience. There are some dependencies which require user input, such as `ICU4XFixedDecimalGroupingStrategy`, which is an enumerator that can be set in the formatter's constructor. Now we have a chain of dependencies and their requirements for use. Some can be constructed in initialization, and some before user input.

Finally, we construct the UI. We noted `ICU4XFixedDecimal` as a required input, so we make sure to show a text field for that. We also noted `ICU4XFixedDecimalGroupingStrategy` as an option for enums, and so we show a dropdown to select these values. We select a default value if the userdoesn't wish to touch it.

All of the behind-the-scenes dependencies and front-facing UI are combined to make an automatically generated webpage for a user to play with. We repeat this process for multiple functions that can be visually represented to the user.

We can create a dropdown or selector to allow easy switching between these visual demos.

This is a human-readable process for creating such a system. Details on how these steps might be accomplished in a more machine-friendly manner are expanded on in Deliverables.

Benefits

Being able to define such a system for automatically creating user interfaces from Rust code has tremendous benefit.

For one, a demo page provides a great way for potential users of ICU4X to immediately see the possibilities in its use. Reluctant users would be able to quickly resolve doubts in seeing the system in action.

For two, an automatic system that requires minimal tweaking leaves room for developers to expand on features without worrying themselves over issues like UI design or accessibility. They would simply need to ensure that whatever authored features are visible to the system for use.

Deliverables

For this project, I propose:

0. A complete detailing of the specifications and systems outlined below.
    a. I expect to diagram the interacting components and their setup for a working proof of concept.
    b. This will take some further investigation to test and throughly examine potential setups, and I estimate a few weeks before it's complete.
1. A re-write of the ICU4X WASM Demo to be automatically generated. Per the example in Background, this requires:
    a. A system for selecting functions, structures, and enumerators to include in the Automatic Demo as visual elements. Additionally, this system would identify dependencies for each visual elements and construction methods for these dependencies.
        i. Most of this system should be automatic, but some user definition will be required to identify features that the FFI does not define. Which functions can be represented visually, for instance.
        ii. I propose writing the majority of this component in Rust, in a fashion very similar to Rust-Diplomat (it could work very easily as a frontend). Adding Rust attributes to FFI items during development, then running a tool to parse the FFI definition AST and generate the requisite WebAssembly for creating UI from FFI definitions.
        iii. Writing this system will most likely take the largest chunk of project time, as designing specifications for how to parse an AST with user experience in mind can get complicated very quickly.
    b. Adapting the existing HTML boilerplate to allow for automatic generation.
        i. This mostly involves continued use of the already in-place dependencies: Webpack, Typescript, SCSS, etc.
        ii. It would just need to ensure compatibility with the automatically generated WebAssembly.
2. An expansion of the ICU4X WASM Demo to include functions beyond those included in the demo.
    a. With the demo re-written to use an automatic system, the next logical step would be expanding to provide the full breadth of features ICU4X has available.
    b. This will most likely include adding attributes to various FFI definitions.
    c. This may also require re-writing the AST parser to accommodate a greater breadth of demo options that the full ICU4X FFI definition provides.

Stretch Goals

1. A re-designed UI.
    a. With expanded example function calls, the UI may require a more efficient way to search and find relevant examples.
    b. A statement explaining the goals of the project and its uses would also help new users orient themselves in exploring ICU4X's possibilities.
    c. Example code sidebar
        i. A sidebar on the webpage which shows how a user could re-create the current FFI selection in the language of their choice.
        ii. In accordance with the benefits described in Benefits, users should be able to see how ICU4X could work practically for them.
2. Dynamically loading language packs as needed.
    a. Speed is always an important factor when it comes to web interfaces, so being able to load in ICU4X's language packs as needed leads to a faster experience overall, and greater accessibility for users with slower internet connections.

Author Experience

Hi, I'm Tyler! I go by ambiguousname on the internet. I'm a senior at University of California, Santa Cruz. I'll be graduating June 2024 with a B.S. in Computer Science: Game Design, and next year I'll be pursuing a M.S. in Applied Mathematics, most likely also at UC Santa Cruz (I'm still in the process of getting responses from grad schools). I have a lot of experience in making tools on the web, and from that comes a whole lot more experience in processing data. There's a lot of different formats in how the web likes to communicate, and I'm pretty good at digging through documentation to learn how the pieces shoudl fit together.

Some selected projects:

- For Rust experience, I'm in the process of re-writing a mod tool I wrote 2 years ago GTK4: https://github.com/ambiguousname/jackbox-custom-content/tree/rust-refactor
- For WebAssembly experience: https://ambiguous.name/2024/02/28/wasm.html
- For HTML/Javascript/Web experience, I wrote a framework for automatically playing a series of HTML games back to back: https://ambiguous.name/2024/01/31/microgame-jam-2022.html
- For making build scripts for automatic HTML page generation, I wrote a Python script that parsed HTML pages into C++ here: https://github.com/ambiguousname/spider-880
- For working with others, I lead a club on-campus in developing games as a team: https://www.gdacollab.com/about-us/. My official title is Chief Operating Officer.
  - I work as a sort of manager of managers, helping officers communicate and tracking important deadlines for us to follow.

I hope this proposal nicely illustrates a dynamic way to construct webpages using ICU4X's existing tools. I look forward to your review, and I thank you for your time.

I hopefully look forward to working with you in the future!