Getting Started with Swift SDKs for WebAssembly

WebAssembly (Wasm) is a virtual instruction set focused on portability, security, and performance. Developers can build client and server applications for Wasm and then deploy them in the browser or other Wasm runtime implementations.

WebAssembly support in Swift started out as a community project. Any instruction set benefits tremendously from a standardized ABI and system interfaces, and from its inception Wasm support in Swift targeted WebAssembly System Interface, which made porting Swift core libraries to this platform much easier.

Starting with Swift 6.2 and development snapshots you can easily cross-compile and run Wasm modules with Swift SDKs for Wasm distributed on swift.org. The distributed artifact bundles also include support for the experimental Embedded Swift mode.

Installation

Note that these steps are required on macOS even if you already have latest Xcode installed. Cross-compilation with Swift SDKs on Windows hosts is not supported yet.

1. Install `swiftly` per the instructions for the platform that you're bulding on.
2. Install Swift 6.2.3 with `swiftly install 6.2.3`.

```
swift sdk install https://download.swift.org/swift-6.2.3-re
```

5. Run `swift sdk list` to verify the Swift SDK was installed and note its ID in the output. Two Swift SDKs will be installed, one with support for all Swift features, and the other with a subset of features allowed in the experimental Embedded Swift mode.

Swift SDK ID	**Description**
`swift-<version>_wasm`	Supports all Swift features
`swift-<version>_wasm-embedded`	Supports a subset of features allowed in the experimental Embedded Swift mode

6. In the future, after installing or selecting a new version of the toolchain with `swiftly` make sure to install and use an exactly matching Swift SDK version.

Building and Running

Let's create a simple package to see the Swift SDK in action:

```
mkdir Hello
cd Hello
swift package init --type executable
```

```
@main
struct wasi_test {
    static func main() {
#if os(WASI)
        print("Hello from WASI!")
#else
        print("Hello from the host system!")
#endif
    }
}
```

Build your package with the following command, substituting the ID from step 5 of the “Installation” section above.

```
swift build --swift-sdk swift-6.2.3-RELEASE_wasm
```

Recent toolchain snapshots that are compatible with Swift SDKs for Wasm also include WasmKit, which is a Wasm runtime that `swift run` can delegate to for execution. To run the freshly built module, use `swift run` with the same `--swift-sdk` option:

```
swift run --swift-sdk swift-6.2.3-RELEASE_wasm
```

You should see the following output:

```
[1/1] Planning build
Building for debugging...
[8/8] Linking Hello.wasm
Build of product 'Hello' complete! (1.31s)
Hello from WASI!
```

Embedded Swift is an experimental subset of the language allowing the toolchain to produce Wasm binaries that are multiple orders of magnitude smaller. One of the Swift SDKs in the artifact bundle you've installed with the `swift sdk install` command is tailored specifically for Embedded Swift.

To build with Embedded Swift SDK, pass its ID as noted in `swift sdk list` output (which has an `-embedded` suffix) in the `--swift-sdk` option. For example:

```
swift build --swift-sdk swift-6.2.3-RELEASE_wasm-embedded
```

or

```
swift run --swift-sdk swift-6.2.3-RELEASE_wasm-embedded
```

Editor Configuration

This section shows you how to configure your development environment for Swift WebAssembly development using the Swift SDKs you installed in the previous section.

Visual Studio Code

If you haven't set up VSCode for Swift development yet, see the Configuring VS Code for Swift Development guide.

Configure VSCode for WebAssembly:

1. Open your Swift package in VSCode.

2.

3. Choose your Swift toolchain from the list (should match the version installed with `swiftly`).

4. When prompted, save the toolchain setting in **Workspace Settings**. This will create or update the `swift.path` setting in `.vscode/settings.json`.

5. Create a `.sourcekit-lsp/config.json` file in your project root:

```
{
    "swiftPM": {
        "swiftSDK": "swift-6.2.3-RELEASE_wasm"
    }
}
```

   Replace `swift-6.2.3-RELEASE_wasm` with the exact Swift SDK ID from your `swift sdk list` output. Use `swift-6.2.3-RELEASE_wasm-embedded` if you're working with Embedded Swift.

6. Reload VSCode using the Command Palette: `Developer: Reload Window`.

Other Editors

For other editors (Vim, Neovim, Emacs, etc.) with LSP support already configured for Swift:

1. Ensure your editor is using the correct Swift toolchain (the one installed with `swiftly`).

2. Create a `.sourcekit-lsp/config.json` file in your project root:

```
        }
    }
```

Replace `swift-6.2.3-RELEASE_wasm` with your Swift SDK ID from `swift sdk list`. Use `swift-6.2.3-RELEASE_wasm-embedded` for Embedded Swift.

For initial Swift LSP setup guides, see:

- Zero to Swift with Neovim
- Zero to Swift with Emacs

Contributed by

Max Desiatov