OpenHarmony

RustNL GOSIM 2024 EUROPE

OHOS OpenHarmony OS for Next Gen Mobile

Jonathan Schwender

Senior OS Engineer

May 6th, 2024

What is OpenHarmony?

Unified ecosystem for apps and services

HarmonyOS

Ultimate experience with software-hardware-chip-cloud integration to support Huawei's high-quality products.

Third-party commercial releases & products

Empower a range of industries.

HUAWEI

Open Source Community & Third-party

OpenHarmony

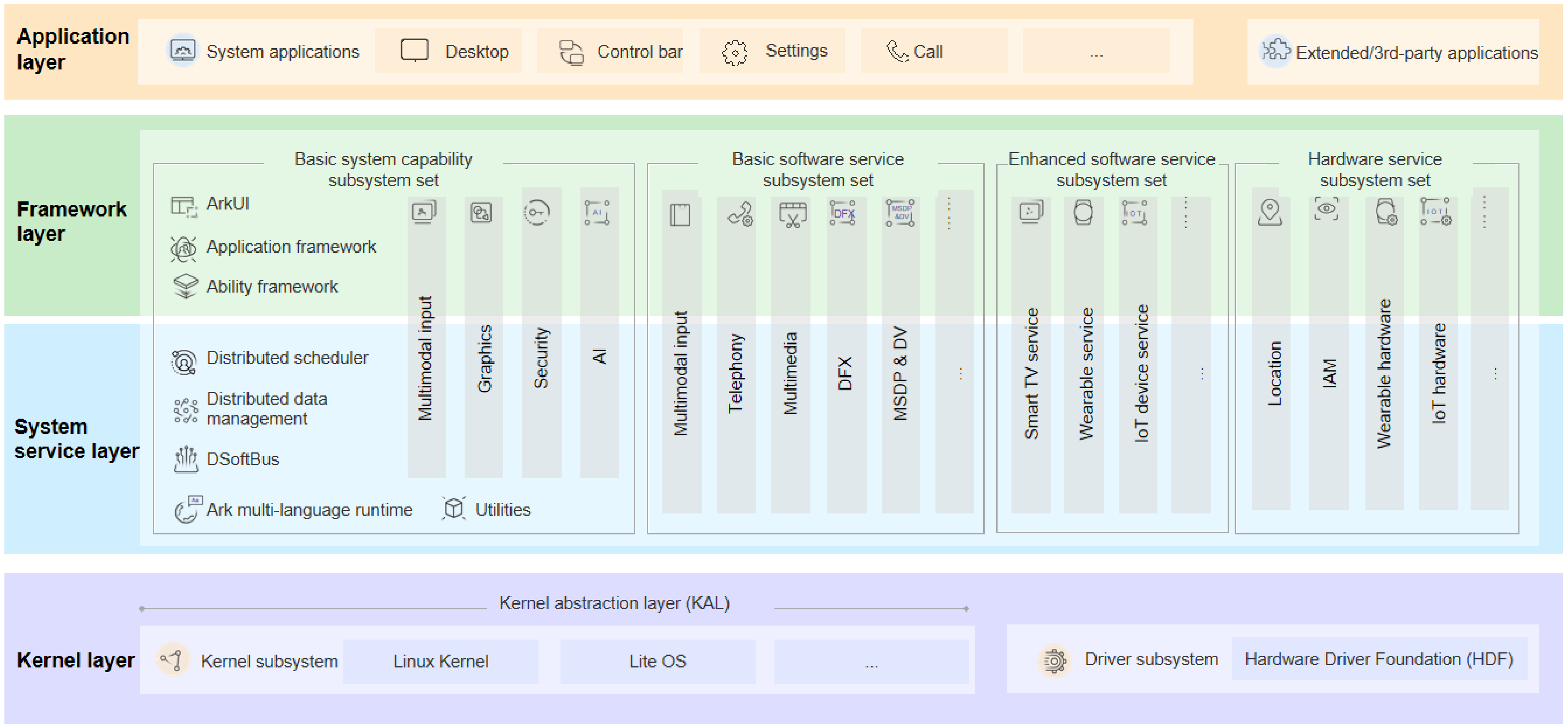
OpenHarmony

Advanced OS base for a connected, intelligent world

Image Source: Chen Haibo, STW 2023

OpenHarmony

OpenHarmony OS

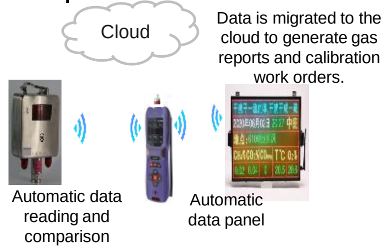


OpenHarmony

350+ Software and Hardware Products Across Key Sectors

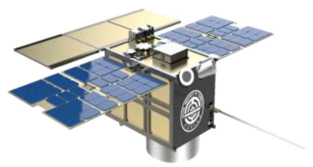
Energy

Mining and electric power terminals


Cloud
Data is migrated to the cloud to generate gas reports and calibration work orders.
Automatic data reading and comparison
Automatic data panel

Aerospace

Satellites



Industry

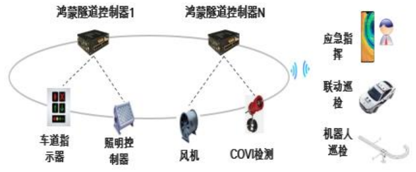
Drones and industrial terminals

Finance

Financial terminals

Transportation

Smart tunnels



Healthcare

Smart medicine cabinets

Education

Harmony classroom

Government

e-Government terminals

Harmony OS NEXT

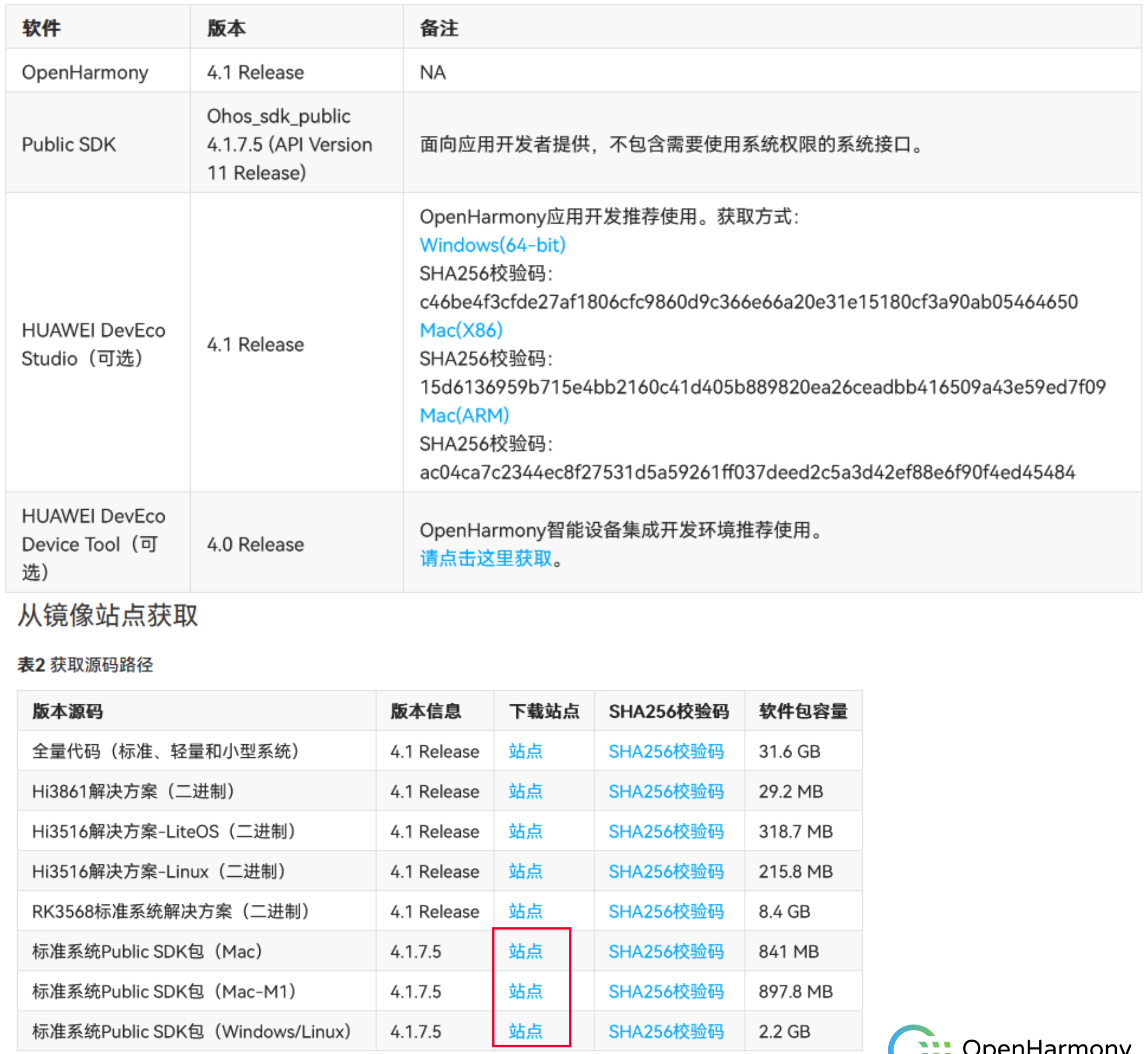
- AOSP (Android) compatibility layer removed
- Apps need to use the new ArkUI framework based on ArkTS (TypeScript)
  - All apps need to be rewritten to use ArkUI
  - Huge effort to port the top 5000 apps to support (Open-) HarmonyOS
- Custom Kernel (with Linux / POSIX compatibility layer)
- Commercial Release： Q4 2024
- Target Audience
  - First: Chinese Mainland

Harmony OS NEXT apps

- Huawei phone users spend 99% of their time in 5000 apps.
- Huge Porting effort
  - **4000** out of the top 5000 apps already ported or being ported
  - Ongoing discussions with the developers of the remaining 1000
- In China Mini-apps are extremely popular
  - Mini web-based apps inside Wechat.
- Many Web developers are already very familiar with TypeScript

DevEco Studio IDE and SDK

- Dev Eco Studio is the official IDE for OpenHarmony
- Latest Release: 4.1
- Dev Eco Studio IDE and the SDK are available from the official release notes
  - English release notes are not available yet.
- Features include:
  - Debugging
  - Hot reloading
  - UI previewer
  - Emulator
  - Profiling / tracing
- Missing: Rust pluging



软件	版本	备注
OpenHarmony	4.1 Release	NA
Public SDK	Ohos_sdk_public 4.1.7.5 (API Version 11 Release)	面向应用开发者提供，不包含需要使用系统权限的系统接口。
HUAWEI DevEco Studio（可选）	4.1 Release	OpenHarmony应用开发推荐使用。获取方式： Windows(64-bit) SHA256校验码： c46be4f3cfde27af1806cfc9860d9c366e66a20e31e15180cf3a90ab05464650 Mac(X86) SHA256校验码： 15d6136959b715e4bb2160c41d405b889820ea26ceadbb416509a43e59ed7f09 Mac(ARM) SHA256校验码： ac04ca7c2344ec8f27531d5a59261ff037deed2c5a3d42ef88e6f90f4ed45484
HUAWEI DevEco Device Tool（可选）	4.0 Release	OpenHarmony智能设备集成开发环境推荐使用。 请点击这里获取。

从镜像站点获取

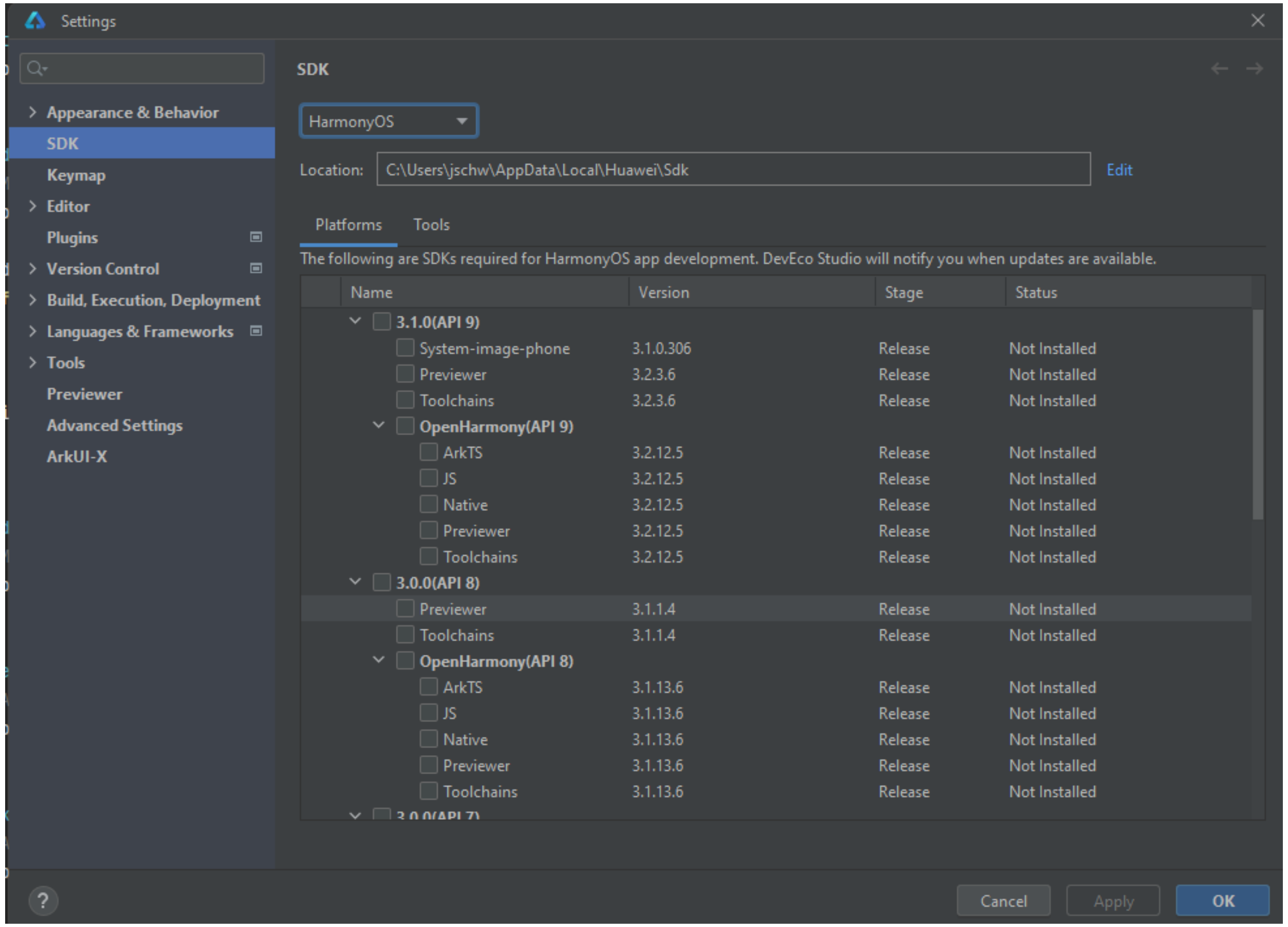
表2 获取源码路径

版本源码	版本信息	下载站点	SHA256校验码	软件包容量
全量代码（标准、轻量和小型系统）	4.1 Release	站点	SHA256校验码	31.6 GB
Hi3861解决方案（二进制）	4.1 Release	站点	SHA256校验码	29.2 MB
Hi3516解决方案-LiteOS（二进制）	4.1 Release	站点	SHA256校验码	318.7 MB
Hi3516解决方案-Linux（二进制）	4.1 Release	站点	SHA256校验码	215.8 MB
RK3568标准系统解决方案（二进制）	4.1 Release	站点	SHA256校验码	8.4 GB
标准系统Public SDK包（Mac）	4.1.7.5	站点	SHA256校验码	841 MB
标准系统Public SDK包（Mac-M1）	4.1.7.5	站点	SHA256校验码	897.8 MB
标准系统Public SDK包（Windows/Linux）	4.1.7.5	站点	SHA256校验码	2.2 GB

OpenHarmony

Dev Eco Studio

- Harmony OS NEXT SDKs are still in closed Developer Preview phase



SDK

HarmonyOS

Location: C:\Users\jschw\AppData\Local\Huawei\Sdk

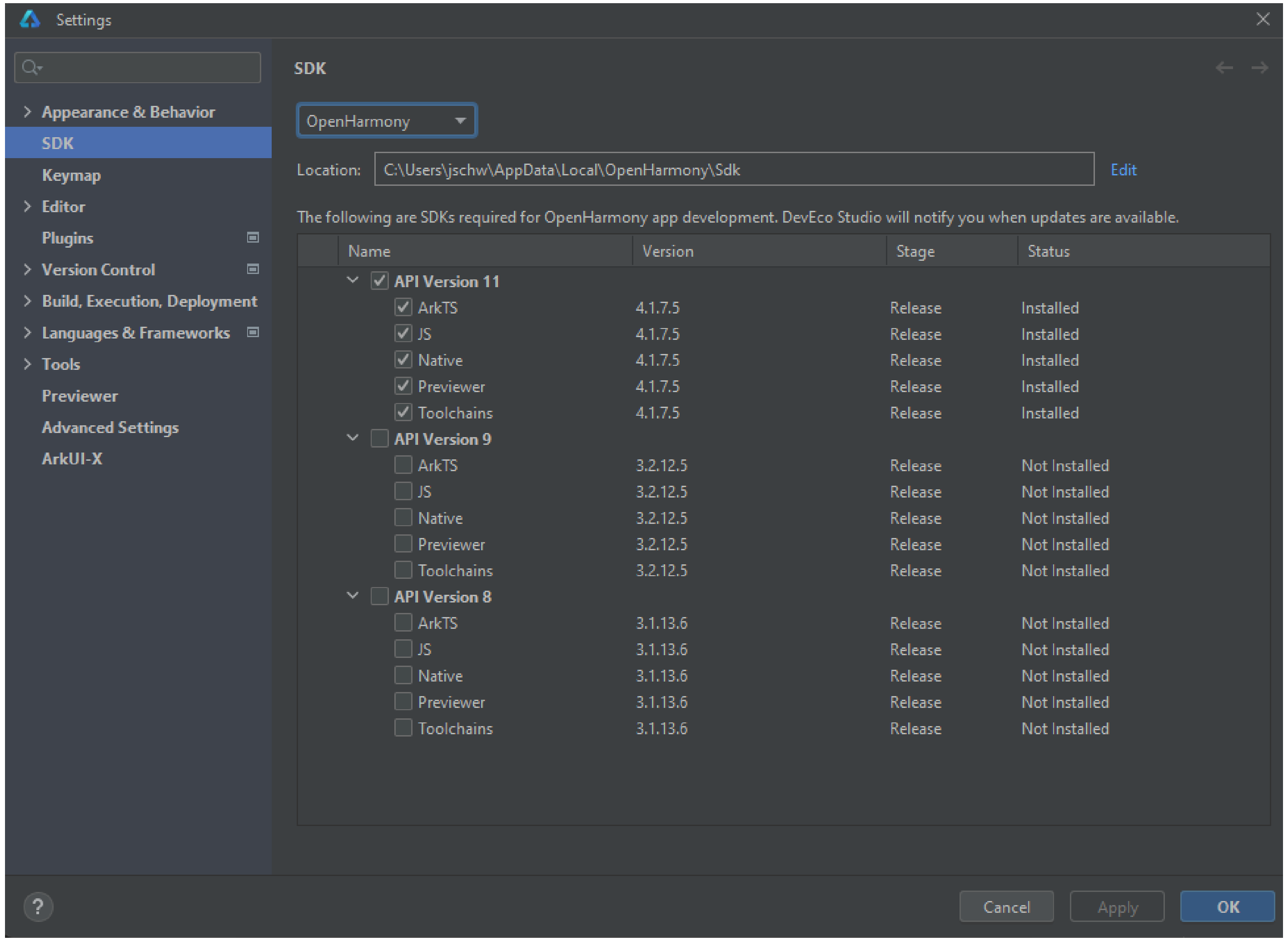
The following are SDKs required for HarmonyOS app development. DevEco Studio will notify you when updates are available.

Name	Version	Stage	Status
3.1.0(API 9)			
System-image-phone	3.1.0.306	Release	Not Installed
Previewer	3.2.3.6	Release	Not Installed
Toolchains	3.2.3.6	Release	Not Installed
OpenHarmony(API 9)			
ArkTS	3.2.12.5	Release	Not Installed
JS	3.2.12.5	Release	Not Installed
Native	3.2.12.5	Release	Not Installed
Previewer	3.2.12.5	Release	Not Installed
Toolchains	3.2.12.5	Release	Not Installed
3.0.0(API 8)			
Previewer	3.1.1.4	Release	Not Installed
Toolchains	3.1.1.4	Release	Not Installed
OpenHarmony(API 8)			
ArkTS	3.1.13.6	Release	Not Installed
JS	3.1.13.6	Release	Not Installed
Native	3.1.13.6	Release	Not Installed
Previewer	3.1.13.6	Release	Not Installed
Toolchains	3.1.13.6	Release	Not Installed
3.0.0(API 7)			

OpenHarmony

Dev Eco Studio

- OpenHarmony 4.1 SDK is freely available
- Can be installed automatically in the IDE
- Additionally select `Native`, if you want to use C/C++/Rust code.



SDK

OpenHarmony

Location: C:\Users\jschw\AppData\Local\OpenHarmony\Sdk Edit

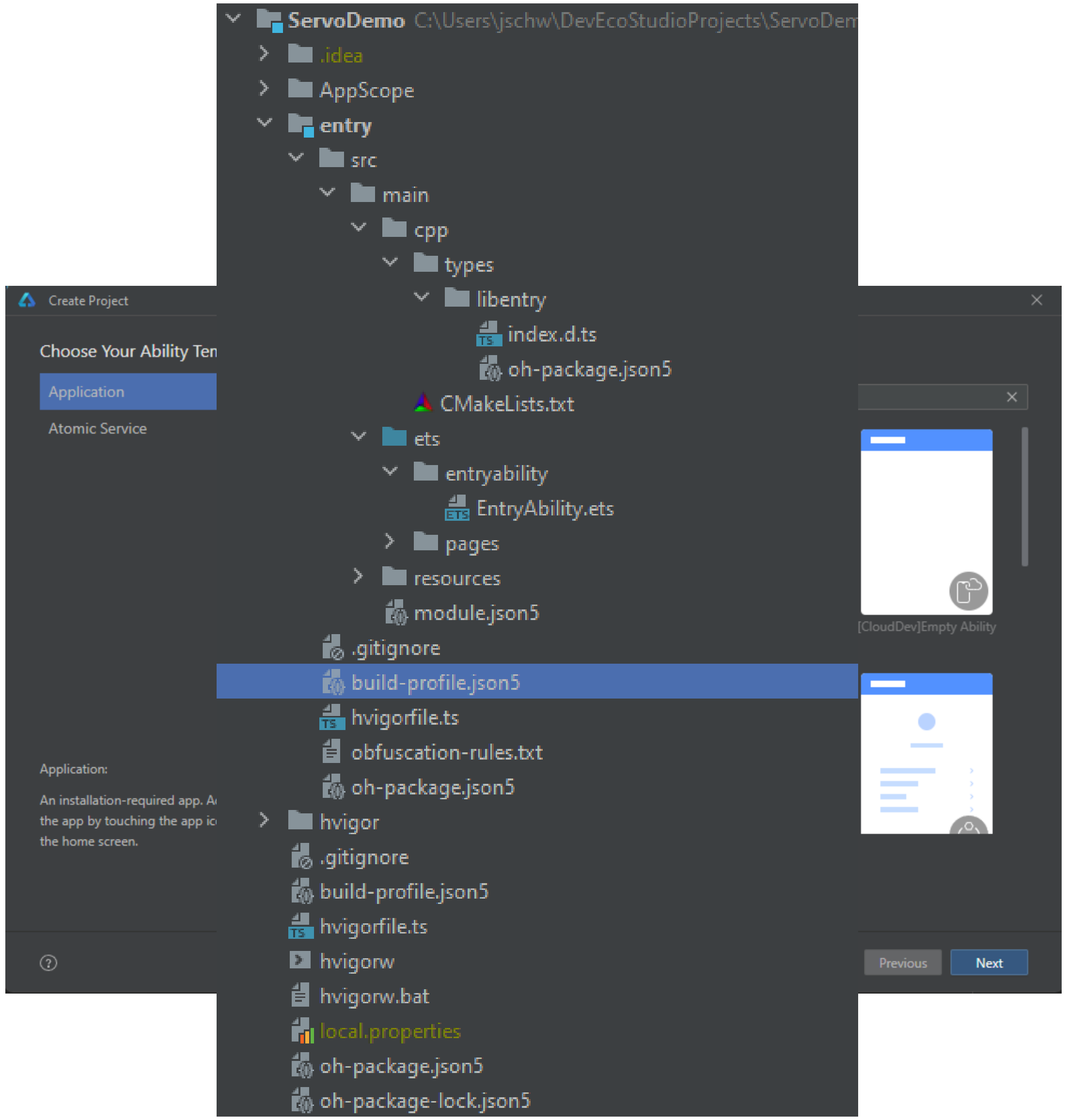
The following are SDKs required for OpenHarmony app development. DevEco Studio will notify you when updates are available.

Name	Version	Stage	Status
✓ API Version 11			
✓ ArkTS	4.1.7.5	Release	Installed
✓ JS	4.1.7.5	Release	Installed
✓ Native	4.1.7.5	Release	Installed
✓ Previewer	4.1.7.5	Release	Installed
✓ Toolchains	4.1.7.5	Release	Installed
API Version 9			
ArkTS	3.2.12.5	Release	Not Installed
JS	3.2.12.5	Release	Not Installed
Native	3.2.12.5	Release	Not Installed
Previewer	3.2.12.5	Release	Not Installed
Toolchains	3.2.12.5	Release	Not Installed
API Version 8			
ArkTS	3.1.13.6	Release	Not Installed
JS	3.1.13.6	Release	Not Installed
Native	3.1.13.6	Release	Not Installed
Previewer	3.1.13.6	Release	Not Installed
Toolchains	3.1.13.6	Release	Not Installed

OpenHarmony

Dev Eco Studio

- Project Wizard to create an app, including all the boilerplate
- Documentation of the package structure
- The build-profile.json5 in the module level configuration contains a `targets` array, where the `runtimeOS` can be set to either HarmonyOS or OpenHarmony.
  > Affects signing of the bundle



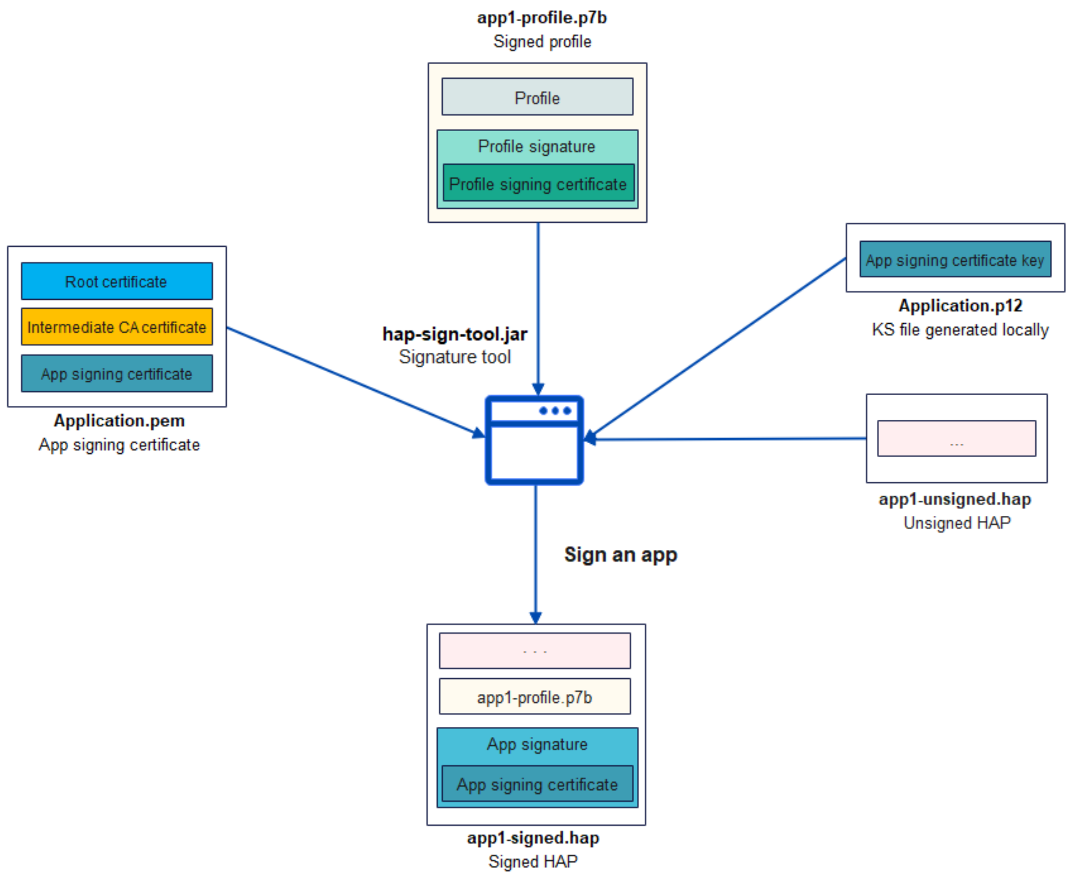
```
ServoDemo C:\Users\jschw\DevEcoStudioProjects\ServoDem
  .idea
  AppScope
  entry
    src
      main
        cpp
          types
            libentry
              index.d.ts
              oh-package.json5
          CMakeLists.txt
        ets
          entryability
            EntryAbility.ets
          pages
        resources
        module.json5
    .gitignore
    build-profile.json5
    hvigorfile.ts
    obfuscation-rules.txt
    oh-package.json5
  hvigor
  .gitignore
  build-profile.json5
  hvigorfile.ts
  hvigorw
  hvigorw.bat
  local.properties
  oh-package.json5
  oh-package-lock.json5
```

Application:

An installation-required app. A the app by touching the app ic the home screen.

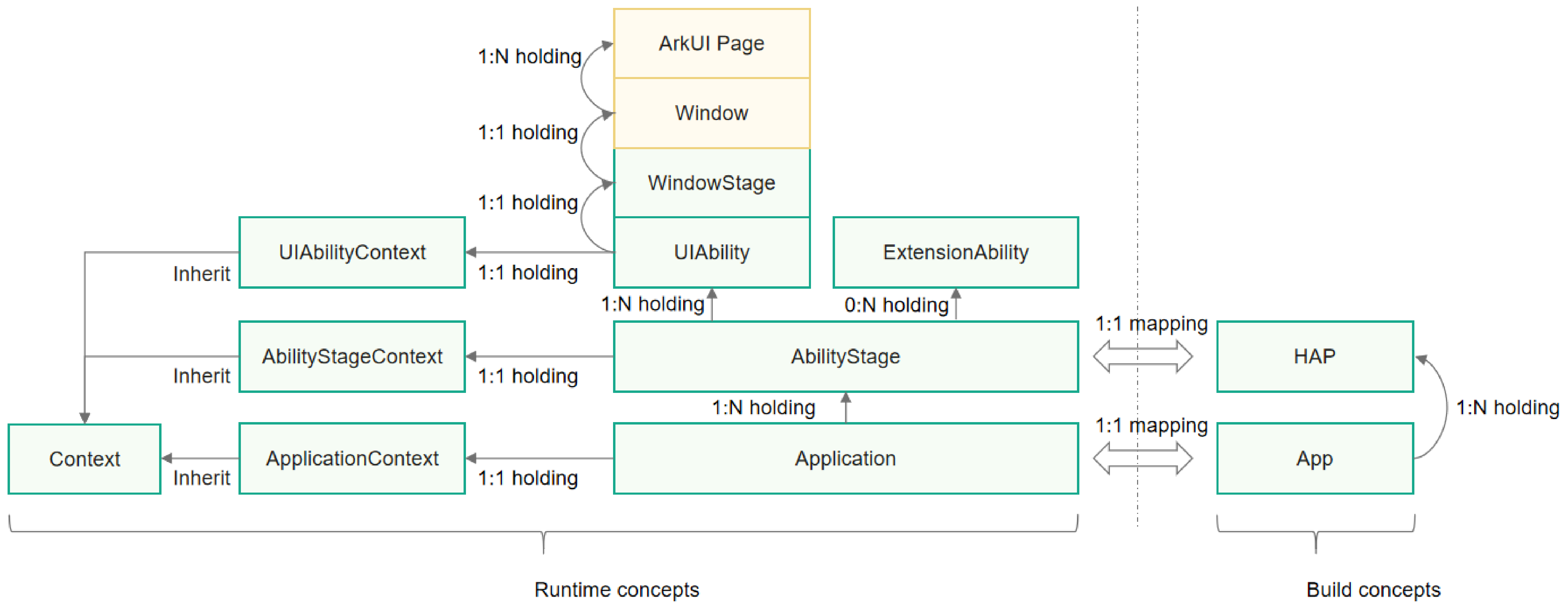
App Signing

- OpenHarmony apps can run on all OpenHarmony devices
- For Security reasons, apps must be signed
- Required signature depends on the OpenHarmony distribution
- Hapsigner tool is used to sign an application bundle
- HarmonyOS: Signing keys can be automatically generated in Dev Eco Studio
- OpenHarmony: Requires manually generating the keys



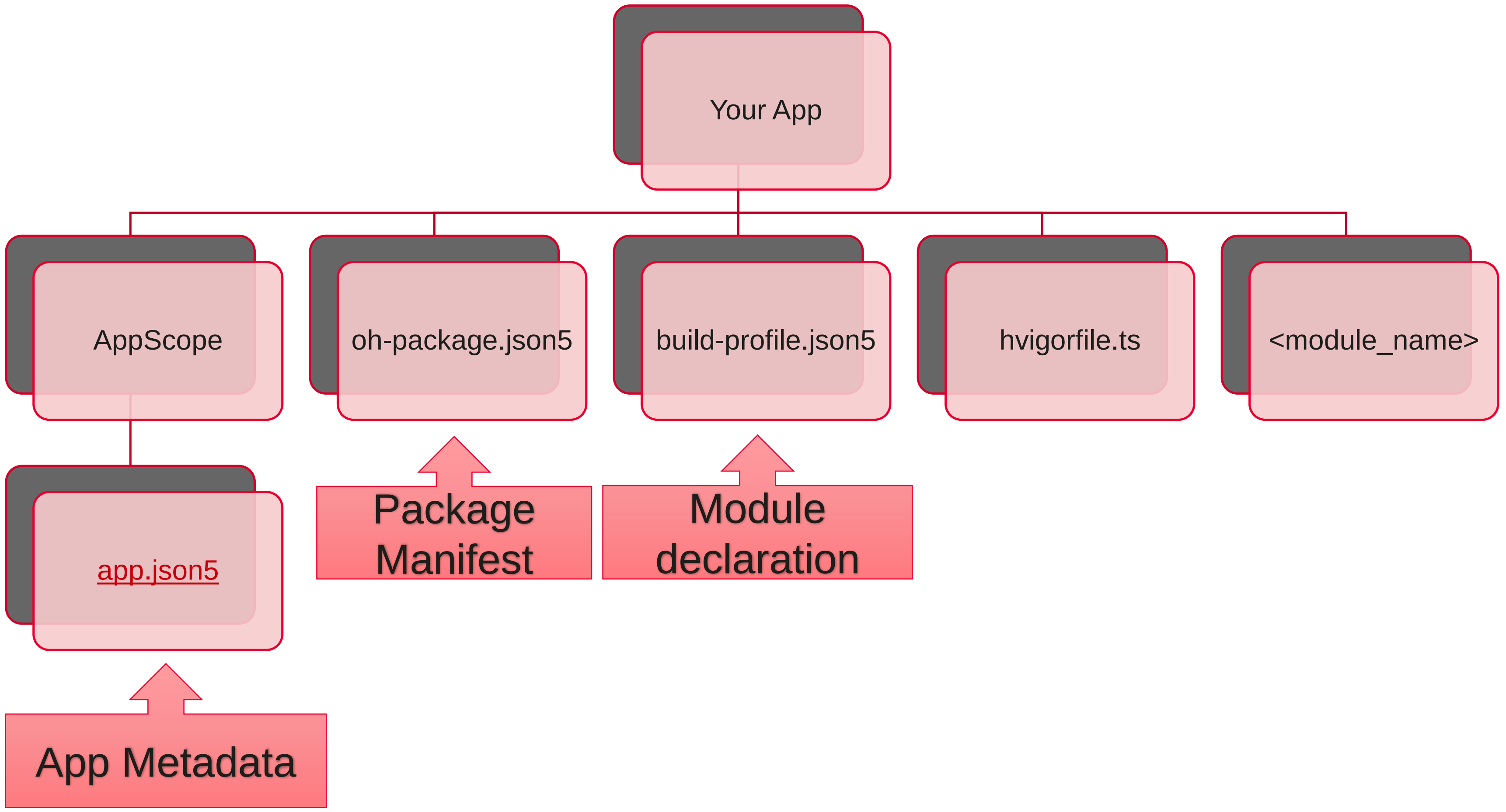
OpenHarmony

OpenHarmony app (Stage Model)

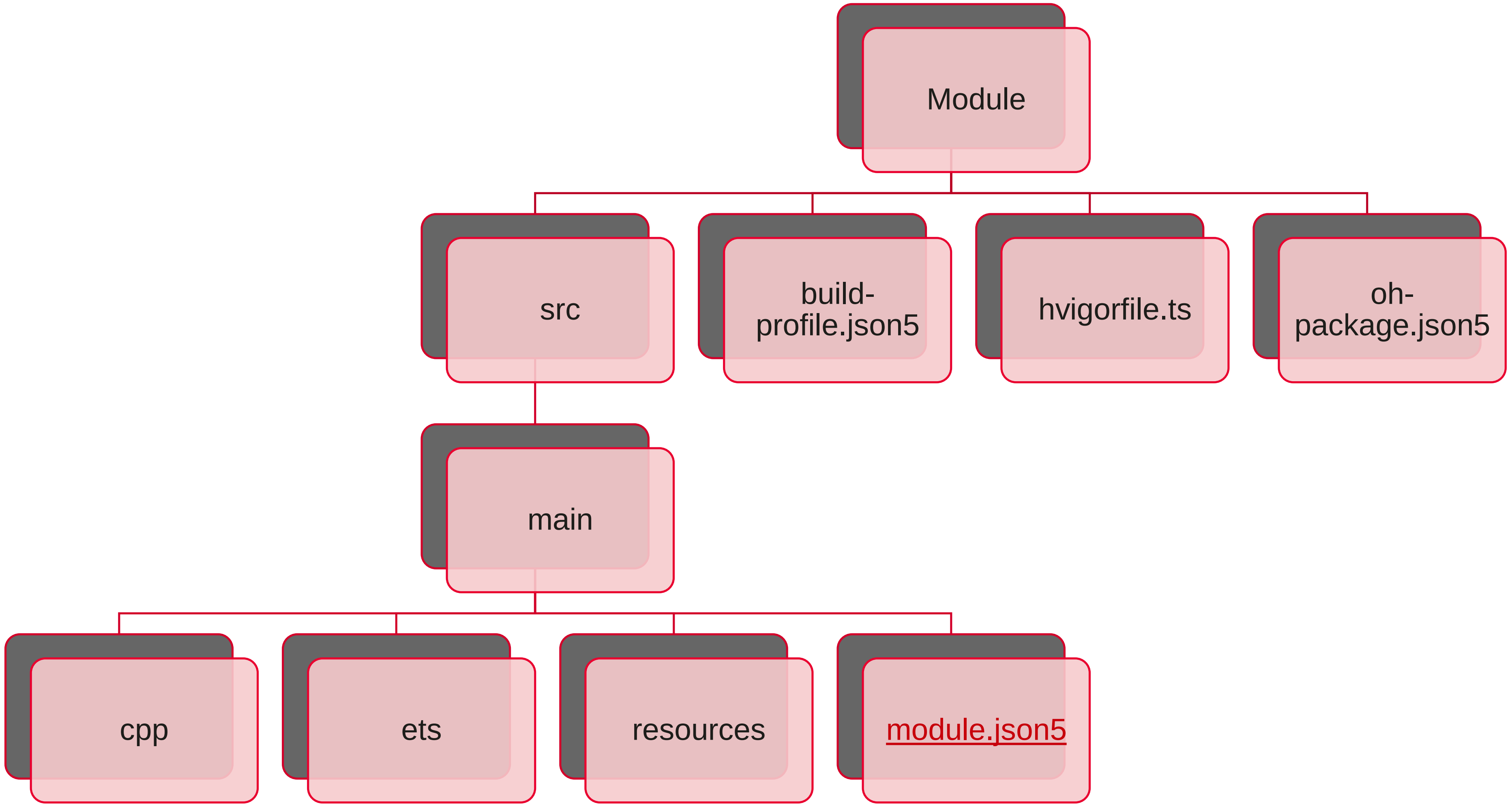


OpenHarmony

Anatomy of an OpenHarmony App

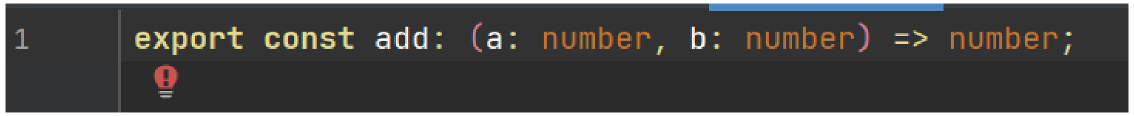


Anatomy of an OpenHarmony app module

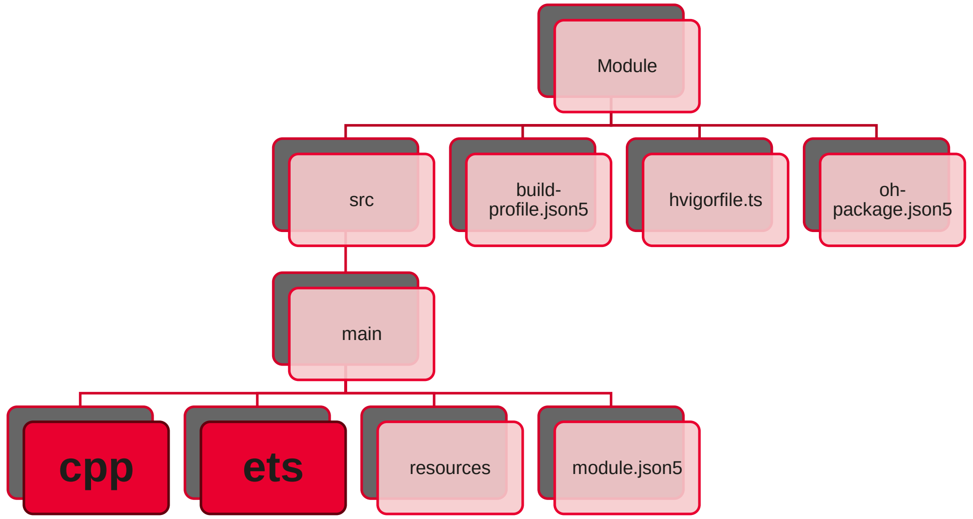


Anatomy of an OpenHarmony app module

- ets: Contains the **Abilities** and **Pages** of the module written in **ArkTS**
  - Commonly: 1x UI Ability with multiple Pages
- **ArkTS** is the primary language for OpenHarmony apps
- cpp: Optional - Native C/C++ code built with CMake
  - Types and Functions are declared via an `index.d.ts` file
  - ArkTS code can import those types / functions



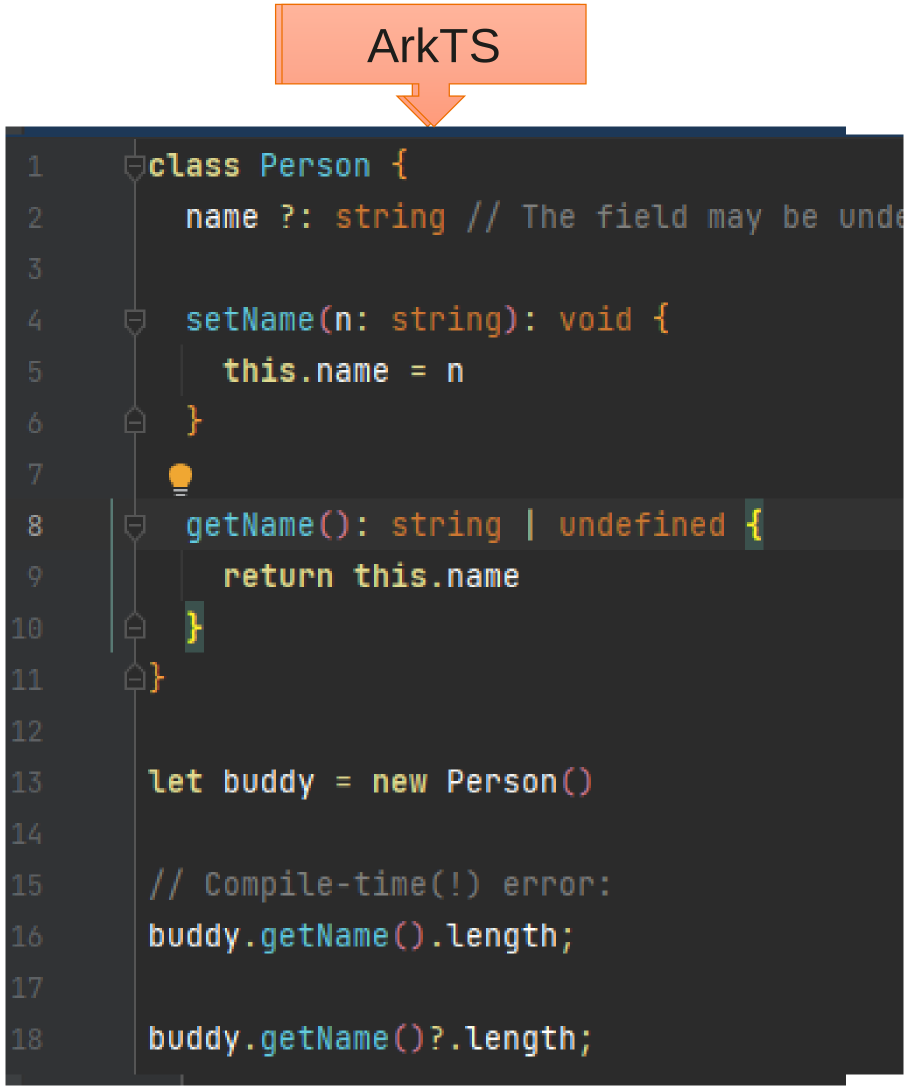
```
export const add: (a: number, b: number) => number;
```



OpenHarmony

ArkTS: Stricter TypeScript flavor

- Goals:
  - > Easy to read
  - > Performance and Efficiency
  - > Prevent common errors
- Static types:
  - > All types are known at compile-time
  - > any/unknown is forbidden
  - > Object layout cannot be changed at runtime
- projects that already follow the best TypeScript practices can keep 90% to 97% of their codebase intact.
- Further reading:
  - > ArkTS introduction
  - > ArkTS migration guide

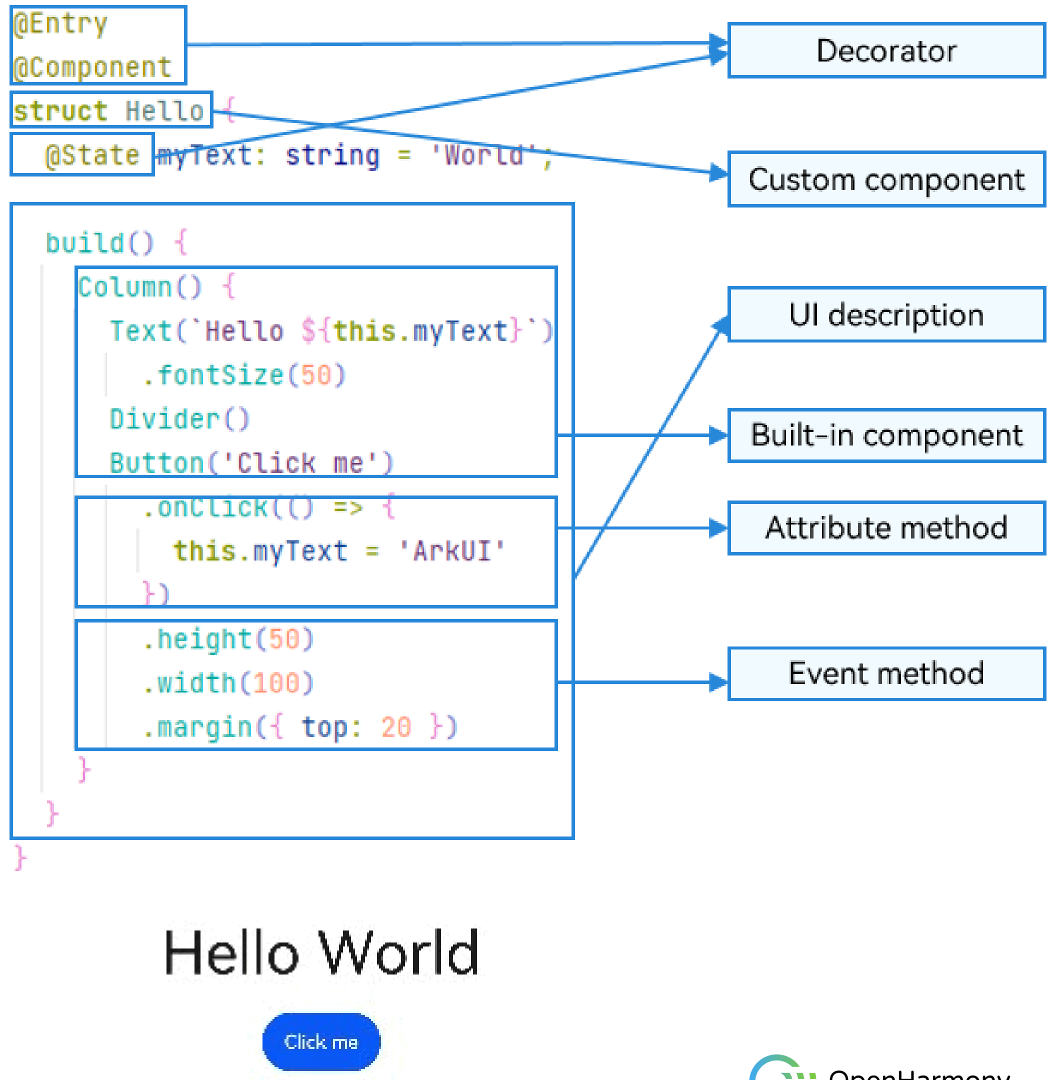


```
class Person {
  name ?: string // The field may be unde
  
  setName(n: string): void {
    this.name = n
  }
  
  getName(): string | undefined {
    return this.name
  }
}

let buddy = new Person()

// Compile-time(!) error:
buddy.getName().length;

buddy.getName()?.length;
```

OpenHarmony

ArkTS – ArkUI specific additions

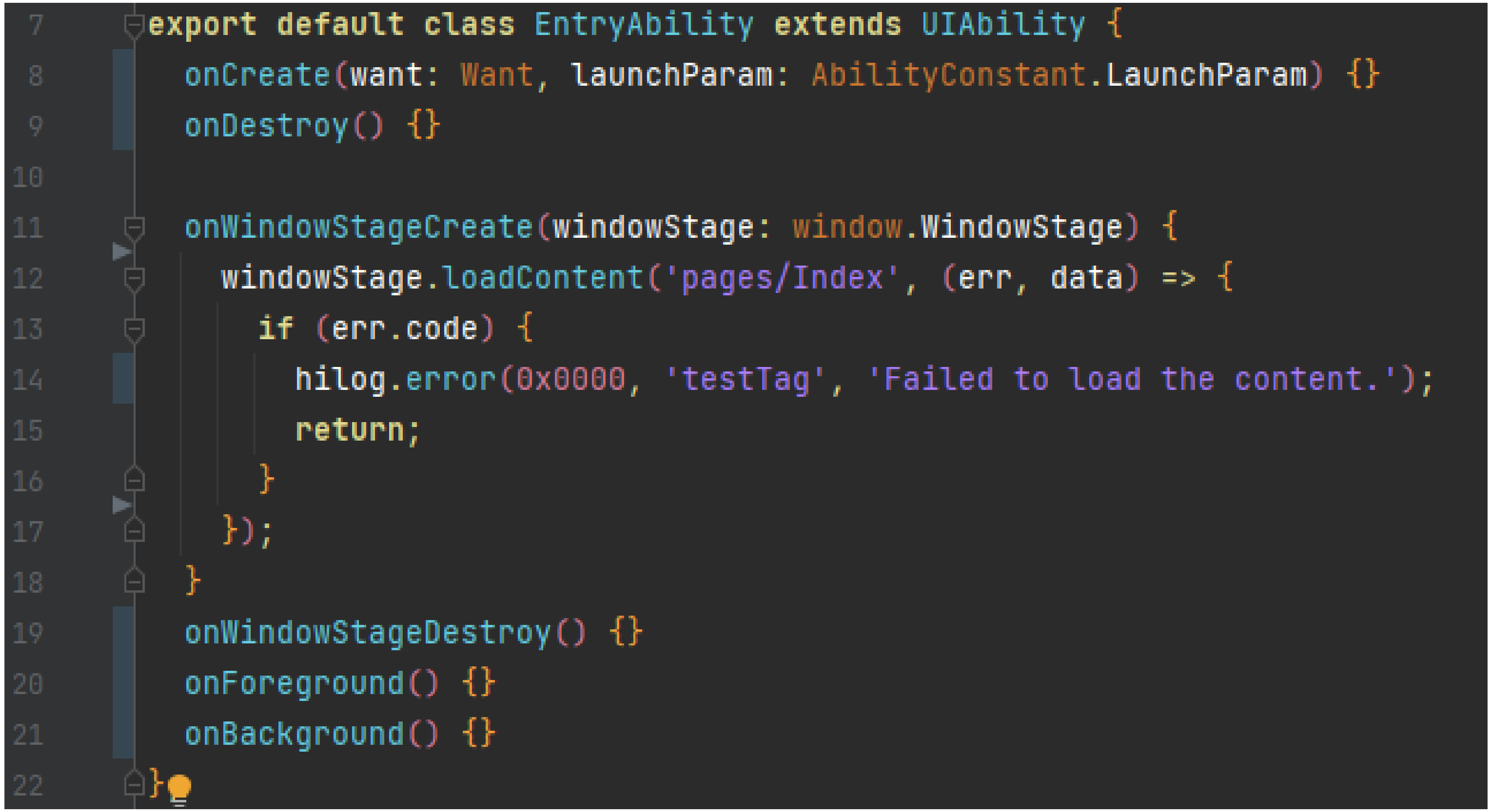
- Additional built-in components
- ArkUI specific decorators
- Used within Pages.



```
@Entry
@Component
struct Hello {
  @State myText: string = 'World';

  build() {
    Column() {
      Text(`Hello ${this.myText}`)
        .fontSize(50)
      Divider()
      Button('Click me')
        .onClick(() => {
          this.myText = 'ArkUI'
        })
        .height(50)
        .width(100)
        .margin({ top: 20 })
    }
  }
}
```

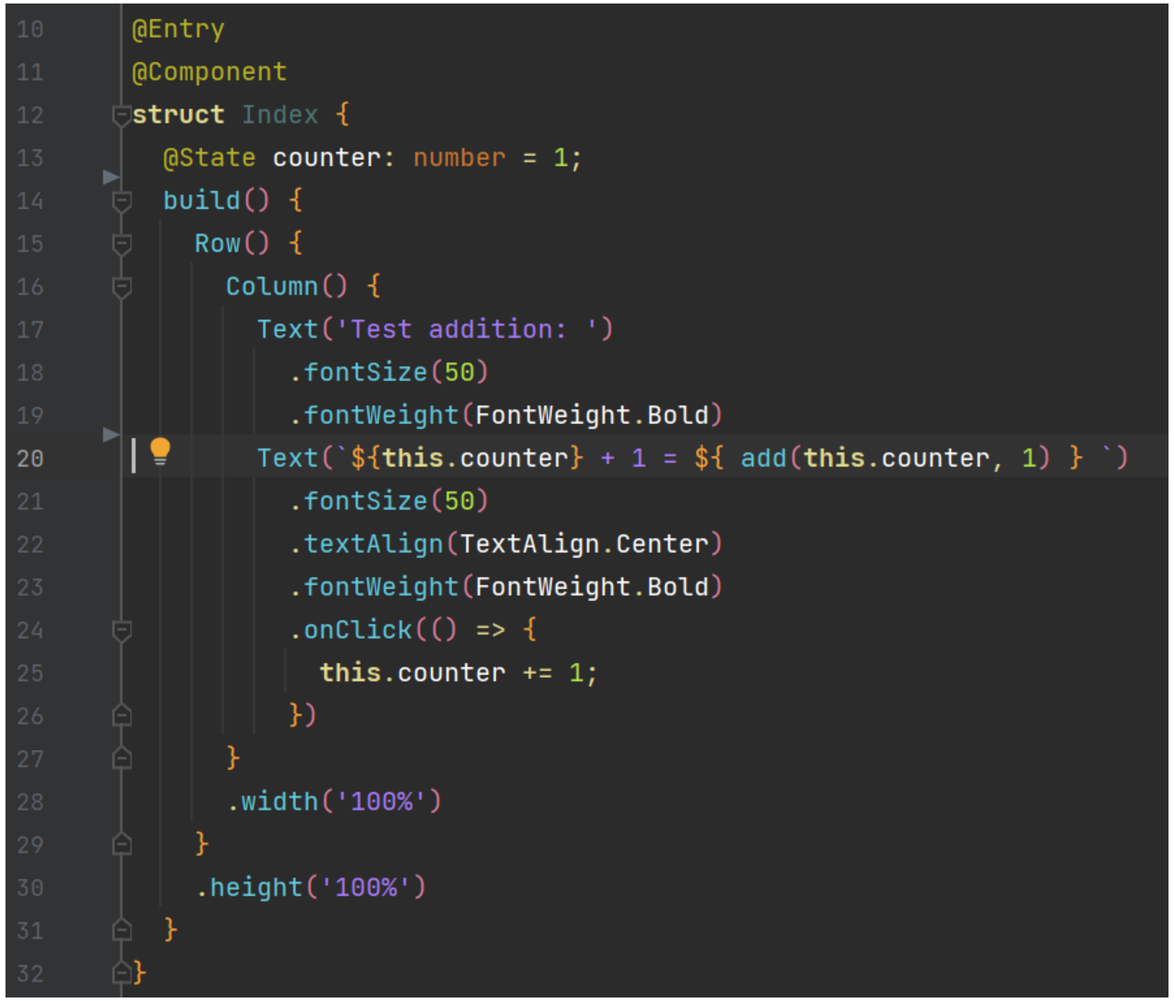
OpenHarmony

Example ArkTS App - Entry Page Ability

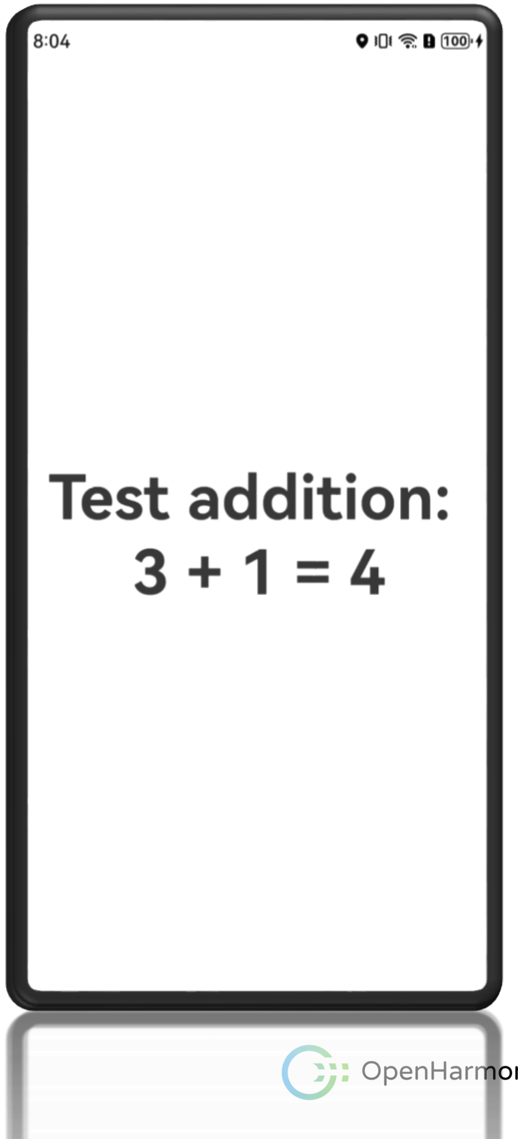


```
export default class EntryAbility extends UIAbility {
  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam) {}
  onDestroy() {}

  onWindowStageCreate(windowStage: window.WindowStage) {
    windowStage.loadContent('pages/Index', (err, data) => {
      if (err.code) {
        hilog.error(0x0000, 'testTag', 'Failed to load the content.');
        return;
      }
    });
  }
  onWindowStageDestroy() {}
  onForeground() {}
  onBackground() {}
}
```

Example: ArkTS App – Main page



```
@Entry
@Component
struct Index {
  @State counter: number = 1;
  build() {
    Row() {
      Column() {
        Text('Test addition: ')
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Text(`${this.counter} + 1 = ${ add(this.counter, 1) } `)
          .fontSize(50)
          .textAlign(TextAlign.Center)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            this.counter += 1;
          })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```



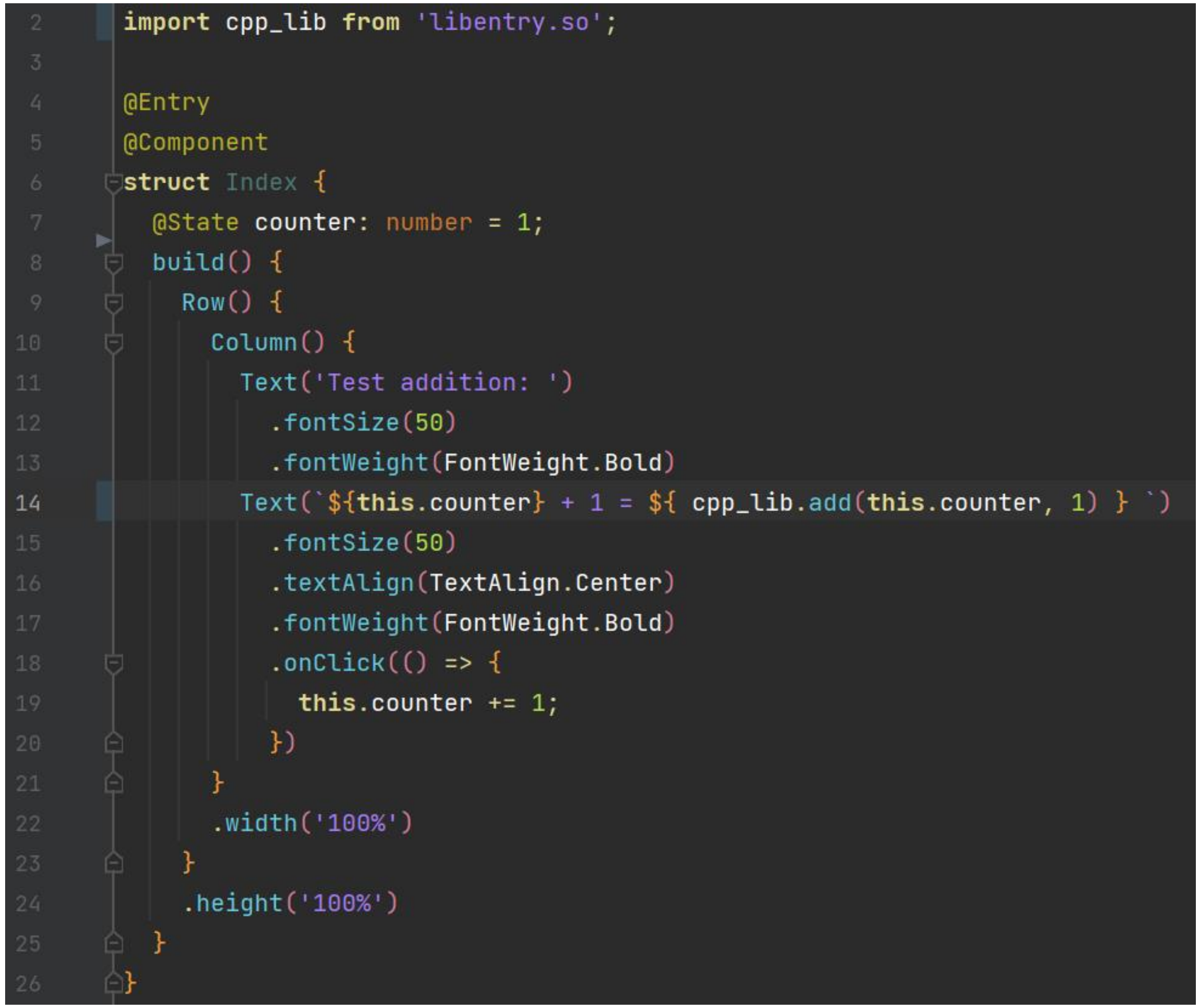
8:04

Test addition:
3 + 1 = 4

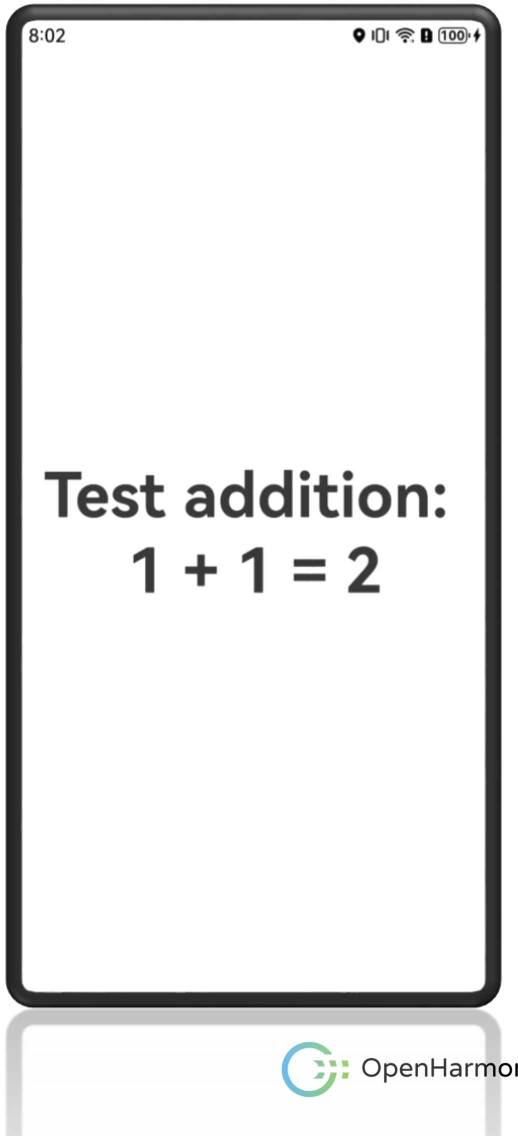
Source-code: https://github.com/jschwe/ohos-rust-demo

OpenHarmony

Example: ArkTS app with native C++ code



```
import cpp_lib from 'libentry.so';

@Entry
@Component
struct Index {
  @State counter: number = 1;
  build() {
    Row() {
      Column() {
        Text('Test addition: ')
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Text(`${this.counter} + 1 = ${ cpp_lib.add(this.counter, 1) } `)
          .fontSize(50)
          .textAlign(TextAlign.Center)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            this.counter += 1;
          })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```



8:02

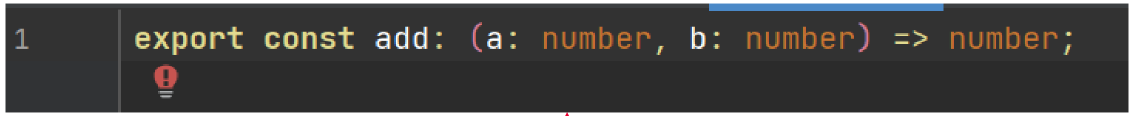
Test addition:
1 + 1 = 2

Source-code: https://github.com/jschwe/ohos-rust-demo

OpenHarmony

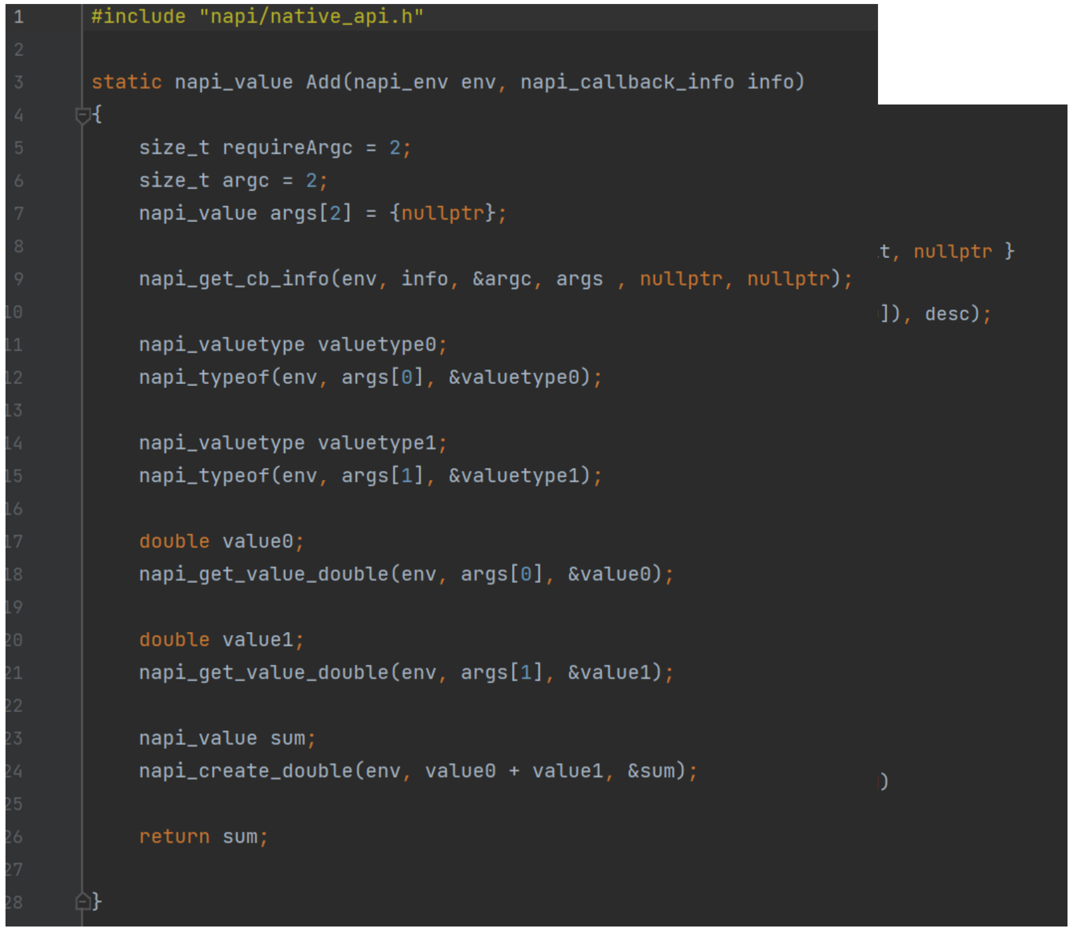
Example app C/C++ Code

- Assumption: `add` takes a long time – We want to speed it up!
- Lots of Boilerplate:
  - module and function registration
  - Extracting the function arguments from the javascript containers
  - Corresponding ArkTS function definition



```
export const add: (a: number, b: number) => number;
```

ArkTS function declaration



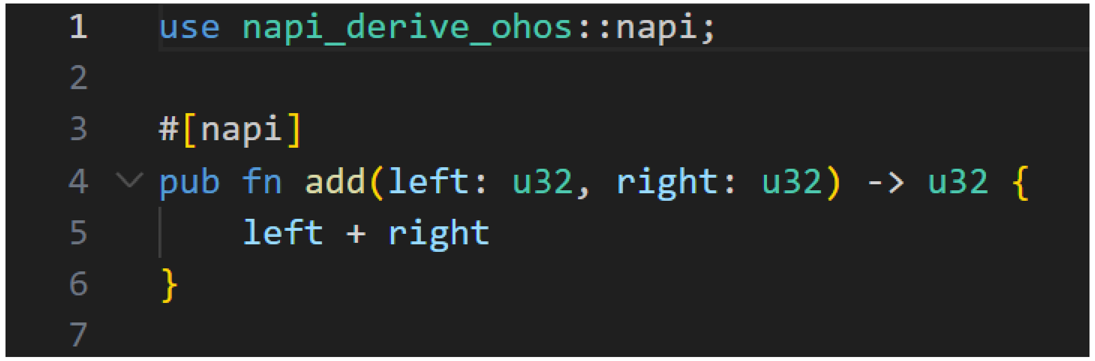
```
#include "napi/native_api.h"

static napi_value Add(napi_env env, napi_callback_info info)
{
    size_t requireArgc = 2;
    size_t argc = 2;
    napi_value args[2] = {nullptr};

    napi_get_cb_info(env, info, &argc, args , nullptr, nullptr);

    napi_valuetype valuetype0;
    napi_typeof(env, args[0], &valuetype0);

    napi_valuetype valuetype1;
    napi_typeof(env, args[1], &valuetype1);

    double value0;
    napi_get_value_double(env, args[0], &value0);

    double value1;
    napi_get_value_double(env, args[1], &value1);

    napi_value sum;
    napi_create_double(env, value0 + value1, &sum);

    return sum;

}
```

```
t, nullptr }
]), desc);
)
```

OpenHarmony

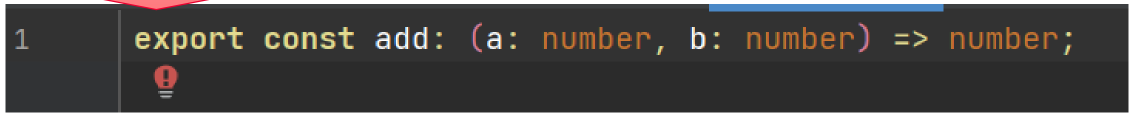
Example app: Rust code

- napi-rs is an existing “framework for building pre-compiled Node.js addons in Rust”
  - Community maintained fork with ohos support `napi-ohos` under development
- Boilerplate is significantly reduced
  - The ArkTS function declaration can be automatically generated by a build-script.



```rust
use napi_derive_ohos::napi;

#[napi]
pub fn add(left: u32, right: u32) -> u32 {
    left + right
}

```

index.d.ts - Declaration



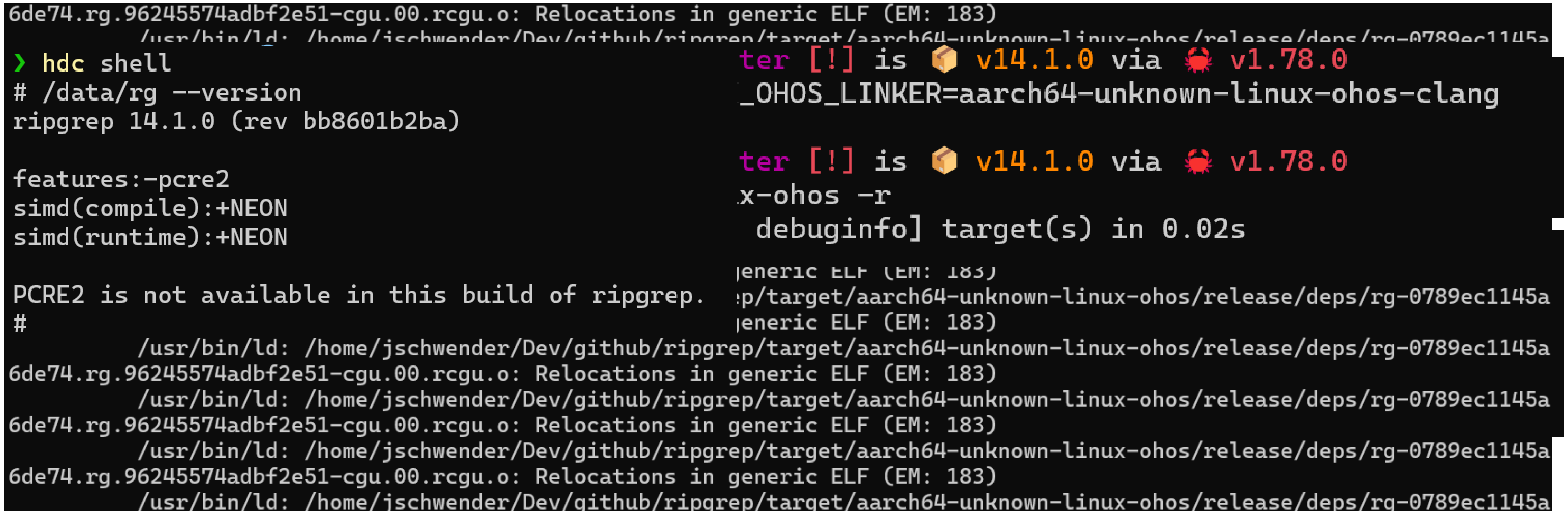
```typescript
export const add: (a: number, b: number) => number;
```

How can we integrate the Rust library?

- A prebuilt dynamic library can be placed under `` `<module_name>/libs/<arch>/lib<name>.so` ``
  - We could setup Dev Eco Studio to build the Rust project and copy the library and the `index.d.ts files.`
- We could write an hvigor plugin in TypeScript
- C/C++ code is built with CMake
- The Corrosion CMake module can automatically import Cargo projects
  - Automatically sets the correct linker and Rust compiler target
  - The OpenHarmony SDK (4.1) ships CMake 3.16, which is missing a required feature
  - The feature could be backported to an older Corrosion version
  - Upstream CMake is missing one file `Platform/OHOS.cmake`
- Conclusion: For now the simplest solution is the first one.

Experiment: Compiling Ripgrep for OHOS

- Popular grep alternative written in Rust
- Add the std library for our target
- We need to specify the linker explicitly



```
6de74.rg.96245574adbf2e51-cgu.00.rcgu.o: Relocations in generic ELF (EM: 183)
        /usr/bin/ld: /home/jschwender/Dev/github/ripgrep/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
```

```
❯ hdc shell
# /data/rg --version
ripgrep 14.1.0 (rev bb8601b2ba)

features:-pcre2
simd(compile):+NEON
simd(runtime):+NEON

PCRE2 is not available in this build of ripgrep.
#
```

```
ter [!] is 📦 v14.1.0 via 🦀 v1.78.0
_OHOS_LINKER=aarch64-unknown-linux-ohos-clang

ter [!] is 📦 v14.1.0 via 🦀 v1.78.0
x-ohos -r
 debuginfo] target(s) in 0.02s
jeneric ELF (EM: 183)
p/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
jeneric ELF (EM: 183)
```

```
        /usr/bin/ld: /home/jschwender/Dev/github/ripgrep/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
6de74.rg.96245574adbf2e51-cgu.00.rcgu.o: Relocations in generic ELF (EM: 183)
        /usr/bin/ld: /home/jschwender/Dev/github/ripgrep/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
6de74.rg.96245574adbf2e51-cgu.00.rcgu.o: Relocations in generic ELF (EM: 183)
        /usr/bin/ld: /home/jschwender/Dev/github/ripgrep/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
6de74.rg.96245574adbf2e51-cgu.00.rcgu.o: Relocations in generic ELF (EM: 183)
        /usr/bin/ld: /home/jschwender/Dev/github/ripgrep/target/aarch64-unknown-linux-ohos/release/deps/rg-0789ec1145a
```

What about bigger, native apps?

- Example: servo, a rendering engine written in Rust
- Main servo components ~240K lines of Rust code
- 700+ Rust and C/C++ dependencies
- Multiple build systems involved
  - cc-rs
  - cmake
  - autotools
- Simple UI (URL bar + Browser window)

Servo Dependencies (Estimation)	
Language	Lines of Code
Rust	3.9 million
C++	1.3 million
C	1.3 million

Counted with \`scc\` on results of \`cargo vendor\`, with winapi*, windows* and ndk crates removed.

Step 1: Compile libservo for OpenHarmony

- Goal: Estimate how much code needs to be adapted to OHOS APIs
- Create a dummy library that depends on libservo and fix all compilation and linking errors

1. Figure out environment variables needed for building C/C++ dependencies (next slide)
   - > Set C/C++-Compilers, sysroot, pkg-config ...
2. Fix Rust dependencies failing to build for OpenHarmony
   - > Often the issue was already fixed by other community members – Just need to update the dependency
   - > But: Updating long dependency chains can be quite time-consuming!
   - > Sometimes backporting an OHOS fix to an older version of a crate can be a quick band-aid solution.
   - > Hardcode / stub everything else that still needs to be implemented (differently) for OpenHarmony
3. All dependencies compile ? -> Fix linking issues
   - > Often simply select feature to build the library from source
   - > Sometimes wrong dependencies get linked in.
     - Example: `the target OS is Linux -> Must have X11 or wayland)

Magic environment variables

- **OHOS_SDK_NATIVE**: Set by Dev Eco Studio to the `native` Directory of the SDK
- `OHOS_LLVM_BIN=${OHOS_SDK_NATIVE}/llvm/bin`
- `CARGO_TARGET_AARCH64_UNKNOWN_LINUX_OHOS_LINKER="${OHOS_LLVM_BIN}/aarch64-unknown-linux-ohos-clang“`
- `PATH=${PATH}:${OHOS_LLVM_BIN}`

Bindgen
`LIBCLANG_PATH=${OHOS_SDK_NATIVE}/llvm/lib`
`CLANG_PATH=${OHOS_LLVM_BIN}/aarch64-unknown-linux-ohos-clang`

Required to avoid bindgen #2682

Magic environment variables Part 2

pkg_config
PKG_CONFIG_SYSROOT_DIR_aarch64_unknown_linux_ohos=${OHOS_SDK_NATIVE}/sysroot
PKG_CONFIG_PATH_aarch64_unknown_linux_ohos="=${OHOS_SDK_NATIVE}/sysroot/usr/lib/pkgconfig:${OHOS_SDK_NATIVE}/sysroot/usr/share/pkgconfig"

cc-rs and cmake-rs
CC_aarch64_unknown_linux_ohos=${OHOS_LLVM_BIN}/aarch64-unknown-linux-ohos-clang
CXX_aarch64_unknown_linux_ohos=${OHOS_LLVM_BIN}/aarch64-unknown-linux-ohos-clang++
AR=${OHOS_LLVM_BIN}/llvm-ar
CXXSTDLIB_aarch64_unknown_linux_ohos=c++
CMAKE_TOOLCHAIN_FILE_aarch64_unknown_linux_ohos=${OHOS_SDK_NATIVE}/build/cmake/ohos.toolchain.cmake
CMAKE_C_COMPILER_aarch64_unknown_linux_ohos=${CC_aarch64_unknown_linux_ohos}
CMAKE_CXX_COMPILER_aarch64_unknown_linux_ohos=${CXX_aarch64_unknown_linux_ohos}

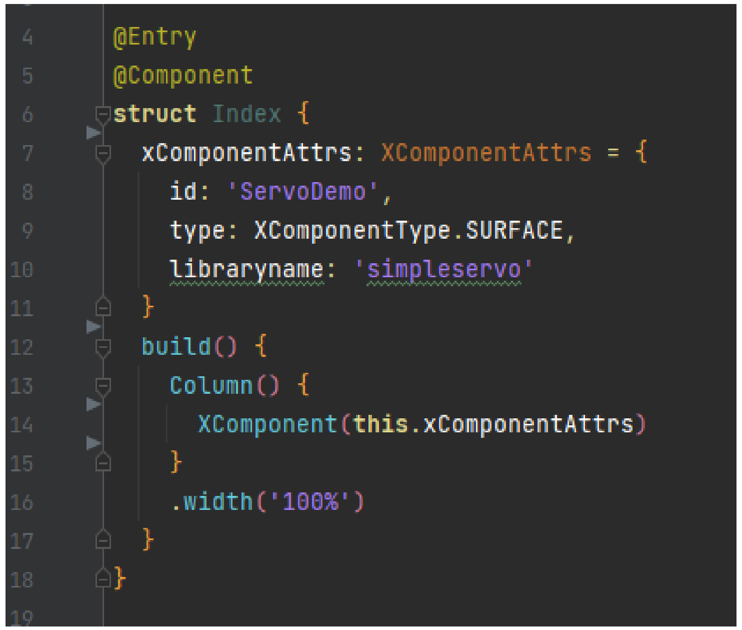
OpenHarmony

Step 2: Create a minimal ArkTS app for libservo

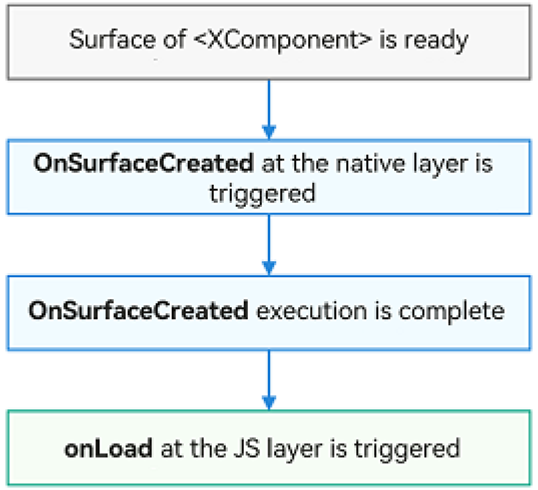
- In Step 1, we „fixed“ some compilation issues by using `unimplemented!()` or `todo!()`.
- Now we implement the missing parts as we hit them.
- But we can still take shortcuts, like hardcoding some values to quickly get a demo.

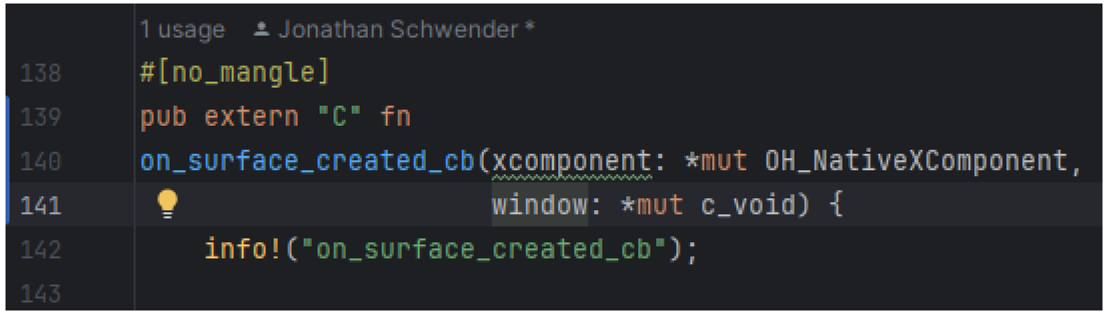
ArkUI XComponent

- XComponent provides a `window` native code can render to
- Start with the simplest possible UI, which is just one Xcomponent
- Initialize servo from there
- Only minor changes required, in platform specific code during the graphics initialization phase
- After that servo loaded and rendered just fine



```
@Entry
@Component
struct Index {
  xComponentAttrs: XComponentAttrs = {
    id: 'ServoDemo',
    type: XComponentType.SURFACE,
    libraryname: 'simpleservo'
  }
  build() {
    Column() {
      XComponent(this.xComponentAttrs)
    }
    .width('100%')
  }
}
```


Surface of <XComponent> is ready
OnSurfaceCreated at the native layer is triggered
OnSurfaceCreated execution is complete
onLoad at the JS layer is triggered

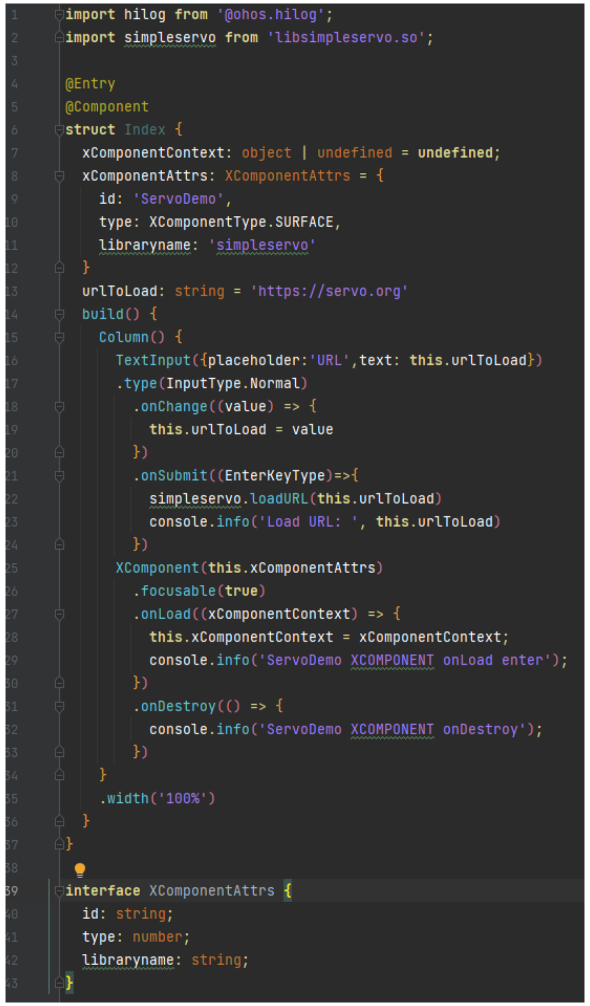


```
#[no_mangle]
pub extern "C" fn
on_surface_created_cb(xcomponent: *mut OH_NativeXComponent,
                                 window: *mut c_void) {
    info!("on_surface_created_cb");
```

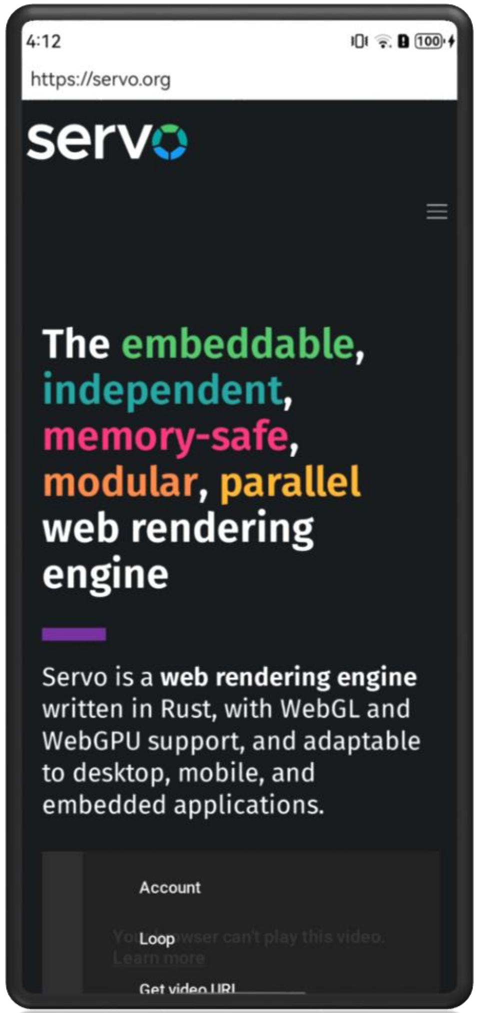
OpenHarmony

Demo: Servo browser app on OpenHarmony

- The UI currently only consists of the URL bar and the browser window
- The bare browsing experience works
- WebGL support is currenlty disabled
- Scrolling works, but fling support is not implemented yet
- Callbacks from Rust to ArkTS are not implemented yet (e.g. Updating the URL bar, if the user clicks a link)
- In the process of upstreaming changes
- Demo ArkUI sources, Servo branch



```
import hilog from '@ohos.hilog';
import simpleservo from 'libsimpleservo.so';

@Entry
@Component
struct Index {
  xComponentContext: object | undefined = undefined;
  xComponentAttrs: XComponentAttrs = {
    id: 'ServoDemo',
    type: XComponentType.SURFACE,
    libraryname: 'simpleservo'
  }
  urlToLoad: string = 'https://servo.org'
  build() {
    Column() {
      TextInput({placeholder:'URL',text: this.urlToLoad})
      .type(InputType.Normal)
        .onChange((value) => {
          this.urlToLoad = value
        })
        .onSubmit((EnterKeyType)=>{
          simpleservo.loadURL(this.urlToLoad)
          console.info('Load URL: ', this.urlToLoad)
        })
      XComponent(this.xComponentAttrs)
        .focusable(true)
        .onLoad((xComponentContext) => {
          this.xComponentContext = xComponentContext;
          console.info('ServoDemo XCOMPONENT onLoad enter');
        })
        .onDestroy(() => {
          console.info('ServoDemo XCOMPONENT onDestroy');
        })
    }
    .width('100%')
  }
}

interface XComponentAttrs {
  id: string;
  type: number;
  libraryname: string;
}
```



4:12

https://servo.org

servo

The embeddable, independent, memory-safe, modular, parallel web rendering engine

Servo is a web rendering engine written in Rust, with WebGL and WebGPU support, and adaptable to desktop, mobile, and embedded applications.

OpenHarmony

Demo: Servo – Changes required

- ArkUI <-> Libservo Layer
  - Easy – Thanks to the trait system
- Adapt OS specific window initialization
  - More challenging, Documentation could be improved
  - Offscreen Buffer still on my todo list
- Adapt the font-loading
- Figure out all the Environment variables that need to be set for the build systems
  - Also depends on the Host OS ...
- Create Rust bindings for OpenHarmony APIs
  - Hilog, Hitrace

Summary: Rust on OpenHarmony

- Since Rust 1.78: `ohos` is supported as a Tier 2 Rust target
  - > Follow the instructions to install the OpenHarmony SDK
  - > Install prebuilt std library via rustup:
  - > `rustup target add aarch64-unknown-linux-ohos`
- The linker should be explicitly set (e.g., via `` `CARGO_TARGET_$TARGET_LINKER` ``)
- Cross-Compiling pure Rust code generally works fine
  - > Some libc functions are purposely not available (1, 2)
- Cross-Compiling code with C/C++ dependencies is a bit more painful
  - > Depending on the involved build-systems a bunch of environment variables need to be set
  - > Some build-systems (autoconf) just fail if they don’t recognize `ohos` and need to be patched.

Future work

- Goal: Make Rust a first-class citizen for native OpenHarmony code
- Provide safe bindings for (more) native OpenHarmony APIs
- Setup a reusable Github CI action
- Explore if the changes required to use the `napi-rs` crates can be merged back upstream
- OpenHarmony provides the Function Flow Runtime Kit (FFRT) for coroutine based scheduling
  - Ideally we would only have one coroutine runtime

Thank you

GOSIM 2024 EUROPE

OpenHarmony