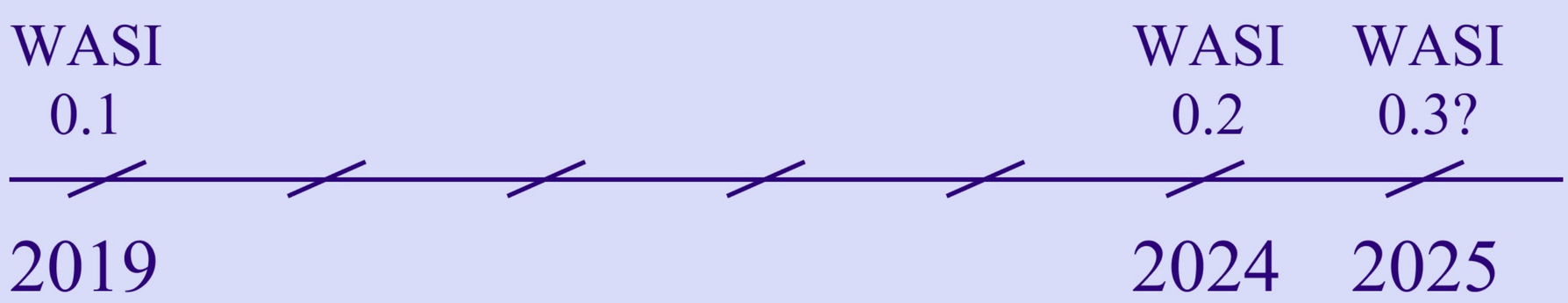
Leveraging the *@since* and *@unstable* gates in WASI

Yosh Wuyts
Microsoft

WASI SG Meeting
2024-05-30

How do we keep evolving WASI between major (breaking) versions?



Fixing bugs

Adding new worlds

Evolving existing worlds

Updating documentation

1. Make it mechanically possible to extend existing WIT documents.

2. Show how we can leverage those mechanics for WASI.

3. Establish a process for releasing new non-breaking WASI versions.

1. Make it mechanically possible to extend existing WIT documents.

2. Show how we can leverage those mechanics for WASI.

3. Establish a process for releasing new non-breaking WASI versions.

@since and *@unstable* make it mechanically possible to extend existing WIT documents.

webassembly/
component-model#332 ←

```
interface foo {
    a: func();

    @since(version = 0.2.1)
    b: func();

    @since(version = 0.2.2, feature = fancy-foo)
    c: func();

    @unstable(feature = fancier-foo)
    d: func();
}
```

```
interface foo {
    a: func();

    @since(version = 0.2.1)
    b: func();

    @since(version = 0.2.2, feature = fancy-foo)
    c: func();

    @unstable(feature = fancier-foo)
    d: func();
}
```

```
interface foo {
    a: func();

    @since(version = 0.2.1)
    b: func();

    @since(version = 0.2.2, feature = fancy-foo)
    c: func();

    @unstable(feature = fancier-foo)
    d: func();
}
```

```
interface foo {
    a: func();

    @since(version = 0.2.1)
    b: func();

    @since(version = 0.2.2, feature = fancy-foo)
    c: func();

    @unstable(feature = fancier-foo)
    d: func();
}
```

```
package ns:p@1.1.0;

interface i {
    @since(version = 1.0.0)
    f: func();

    @since(version = 1.1.0)
    g: func();
}
```

Target version is
1.0.0 →

```
(component
  (type (export "i") (component
    (export "ns:p/i@1.0.0" (instance
      (export "f" (func))
    ))
  ))
)
```

Target version is
1.1.0 →

```
(component
  (type (export "i") (component
    (export "ns:p/i@1.1.0" (instance
      (export "f" (func))
      (export "g" (func))
    ))
  ))
)
```

@since and *@unstable* are not part of the runtime semantics of components, they are part of the source-level tooling for producing components.

End of the First Presentation

1. Make it mechanically possible to extend existing WIT documents.

2. Show how we can leverage those mechanics for WASI.

3. Establish a process for releasing new non-breaking WASI versions.

```
package wasi:clocks@0.2.0;

interface timezone {
    use wall-clock.{datetime};

    display: func(when: datetime) -> timezone-display;

    utc-offset: func(when: datetime) -> s32;

    record timezone-display {
        utc-offset: s32,
        name: string,
        in-daylight-saving-time: bool,
    }
}
```

Extensions to existing specifications need to independently advance through the phase process, starting with stage 0.

webassembly/
wasi#605 ←

```
package wasi:clocks@0.2.0;

@unstable(feature = clocks-timezone)
interface timezone {
    use wall-clock.{datetime};

    @unstable(feature = clocks-timezone)
    display: func(when: datetime) -> timezone-display;

    @unstable(feature = clocks-timezone)
    utc-offset: func(when: datetime) -> s32;

    @unstable(feature = clocks-timezone)
    record timezone-display {
        utc-offset: s32,
        name: string,
        in-daylight-saving-time: bool,
    }
}
```

```
package wasi:clocks@0.2.1;

@since(version = 0.2.1, feature = clocks-timezone)
interface timezone {
    use wall-clock.{datetime};

    @since(version = 0.2.1, feature = clocks-timezone)
    display: func(when: datetime) -> timezone-display;

    @since(version = 0.2.1, feature = clocks-timezone)
    utc-offset: func(when: datetime) -> s32;

    @since(version = 0.2.1, feature = clocks-timezone)
    record timezone-display {
        utc-offset: s32,
        name: string,
        in-daylight-saving-time: bool,
    }
}
```

```
package wasi:clocks@0.2.x;

@since(version = 0.2.1)
interface timezone {
    use wall-clock.{datetime};

    @since(version = 0.2.1)
    display: func(when: datetime) -> timezone-display;

    @since(version = 0.2.1)
    utc-offset: func(when: datetime) -> s32;

    @since(version = 0.2.1)
    record timezone-display {
        utc-offset: s32,
        name: string,
        in-daylight-saving-time: bool,
    }
}
```

Existing proposals will benefit from versioning their existing API surface.

webassembly/
wasi#604 ←

```
package wasi:random@0.2.0;

/// WASI Random is a random data API.
interface random {
    /// Return `len` cryptographically-secure random or
pseudo-random bytes.
    get-random-bytes: func(len: u64) -> list<u8>;

    /// Return a cryptographically-secure random or
pseudo-random `u64` value.
    get-random-u64: func() -> u64;
}
```

```
package wasi:random@0.2.0;

/// WASI Random is a random data API.
@since(version = 0.2.0)
interface random {
    /// Return `len` cryptographically-secure random or
pseudo-random bytes.
    @since(version = 0.2.0)
    get-random-bytes: func(len: u64) -> list<u8>;

    /// Return a cryptographically-secure random or
pseudo-random `u64` value.
    @since(version = 0.2.0)
    get-random-u64: func() -> u64;
}
```

```
package wasi:random@0.3.0;

/// WASI Random is a random data API.
@since(version = 0.2.0)
interface random {
    /// Return `len` cryptographically-secure random or
pseudo-random bytes.
    @since(version = 0.2.0)
    get-random-bytes: func(len: u64) -> list<u8>;

    /// Return a cryptographically-secure random or
pseudo-random `u64` value.
    @since(version = 0.2.0)
    get-random-u64: func() -> u64;
}
```

Evolving APIs requires care, especially when working with re-exports.

webassembly/
wasi-http#115 ←

```
package wasi:http@0.2.0;

world imports {
  include wasi:clocks/imports@0.2.0;

  import wasi:random/random@0.2.0;

  import wasi:cli/stdout@0.2.0;
  import wasi:cli/stderr@0.2.0;
  import wasi:cli/stdin@0.2.0;

  import outgoing-handler;
}
```

```
package wasi:clocks@0.2.1;

@since(version = 0.2.0)
world imports {
  @since(version = 0.2.0)
  import monotonic-clock;
  @since(version = 0.2.0)
  import wall-clock;

  @since(version = 0.2.1)
  import timezone;
}
```

```
package wasi:http@0.2.1;

world imports {
  include wasi:clocks/imports@0.2.0;

  import wasi:random/random@0.2.1;

  import wasi:cli/stdout@0.2.1;
  import wasi:cli/stderr@0.2.1;
  import wasi:cli/stdin@0.2.1;

  import outgoing-handler;
}
```

```
package wasi:http@0.2.1;

world imports {
  import wasi:clocks/monotonic-clock@0.2.1;
  import wasi:clocks/wall-clock@0.2.1;

  import wasi:random/random@0.2.1;

  import wasi:cli/stdout@0.2.1;
  import wasi:cli/stderr@0.2.1;
  import wasi:cli/stdin@0.2.1;

  import outgoing-handler;
}
```

1. Make it mechanically possible to extend existing WIT documents.

2. Show how we can leverage those mechanics for WASI.

3. Establish a process for releasing new non-breaking WASI versions.

End of the Second Presentation

Thanks to

Alex Crichton - *Fermyon*

Bailey Hayes - *Cosmonic*

Dan Gohman - *Fastly*

Luke Wagner - *Fastly*