A Modern IEEE 2030.5 Client Implementation

Thesis B - Trimester 2, 2023

By Ethan Dickson (z5309251)

Supervised by Jawad Ahmed

Assessed by Nadeem Ahmed

School of Computer Science & Engineering
University of New South Wales, Sydney

Outline

Progress:

Check-In

April 26th to May 29th, 2023

- Common Library: Resource Serialization & Deserialization Test Suite ✓
- Common Library: Cipher Suite Interface + Unit Tests (Security FS) ✓
- Common Library: Network Interface + Unit Tests (Application Support FS) ✓
- Mock IEEE 2030.5 Server & System Test Framework ✓
- Test Client Binary ✓
- Client Library: Event-Based Architecture ✓

Progress:

Check-In

May 29th to September 11th, 2023

- Client Library
  - Time FS: Scheduled Events, Server polling ✗
  - End Device FS: Client & Server Relationship ✓
  - Subscription/Notification Resource retrieval ✓
  - Function Sets: Metering, LogEvent, FileDownload, Demand Response and Load Control, Distributed Energy Resources ✓
  - Corresponding Unit Tests ✓
- Mock Server Updates ✓
- Test Client Updates ✓
- Thesis B Demonstration of all implemented functionality ✓

Progress:

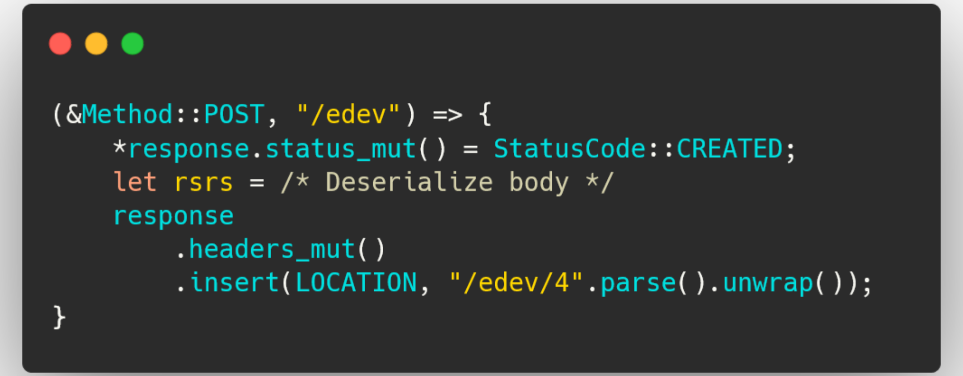
2030.5 Security in Rust

- We are required to use the ‘ECDHE-ECDSA-AES128-CCM8’ cipher suite.
    - Currently, no native Rust implementation
- Implemented in `openssl`, of which there are Rust bindings for.
- `hyper` HTTP library + `hyper_openssl`
- New Stretch Goal: Implement cipher suite in `rustTLS`
- Real Certificate Authorities: Can’t use loopbacks.
- Self-Signed Certificates: Trust issues.
- `mkCert` - Manage our own CA, generate certs.

Progress:

2030.5 Test Server

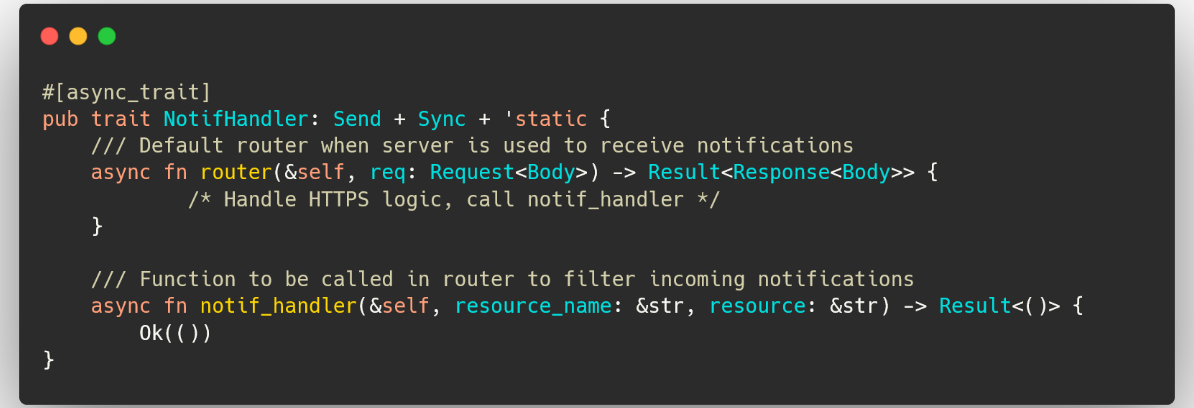
- We require a server hosting resources to test our client.
- Subscription / Notification Mechanism – Client operates as a server
- `tokio_openssl` allows us to do SSL asynchronously.
- `hyper` - Generic server interface.
- Existing Server – Bare Minimum



```
(&Method::POST, "/edev") => {
    *response.status_mut() = StatusCode::CREATED;
    let rsrs = /* Deserialize body */
    response
        .headers_mut()
        .insert(LOCATION, "/edev/4".parse().unwrap());
}
```

Progress:
Client Subscription / Notification

- Abstraction over a 2030.5 Server.
    - Create subscriptions, routes, callbacks for specific notifications.
- Current Implementation:



```rust
#[async_trait]
pub trait NotifHandler: Send + Sync + 'static {
    /// Default router when server is used to receive notifications
    async fn router(&self, req: Request<Body>) -> Result<Response<Body>> {
            /* Handle HTTPS logic, call notif_handler */
    }

    /// Function to be called in router to filter incoming notifications
    async fn notif_handler(&self, resource_name: &str, resource: &str) -> Result<()> {
        Ok(())
}
```

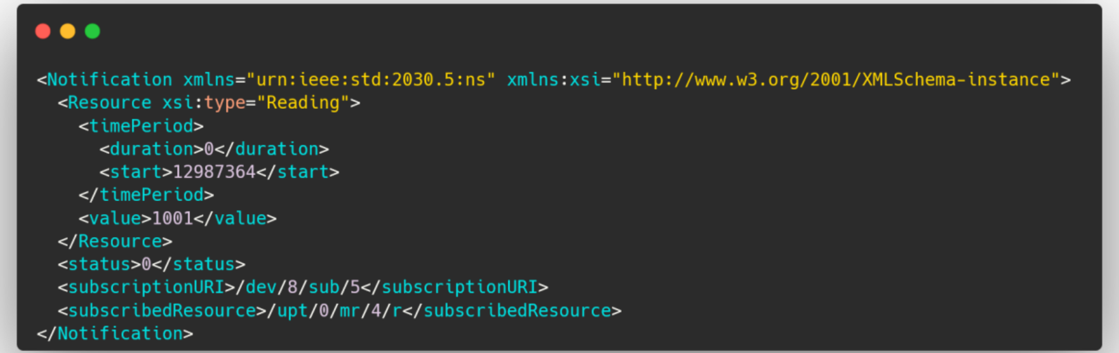
Progress:
Client Polling

- Automated, Scheduled GET requests with callbacks.
- To be completed between T2 and T3.
- Requires non-global state in callbacks.

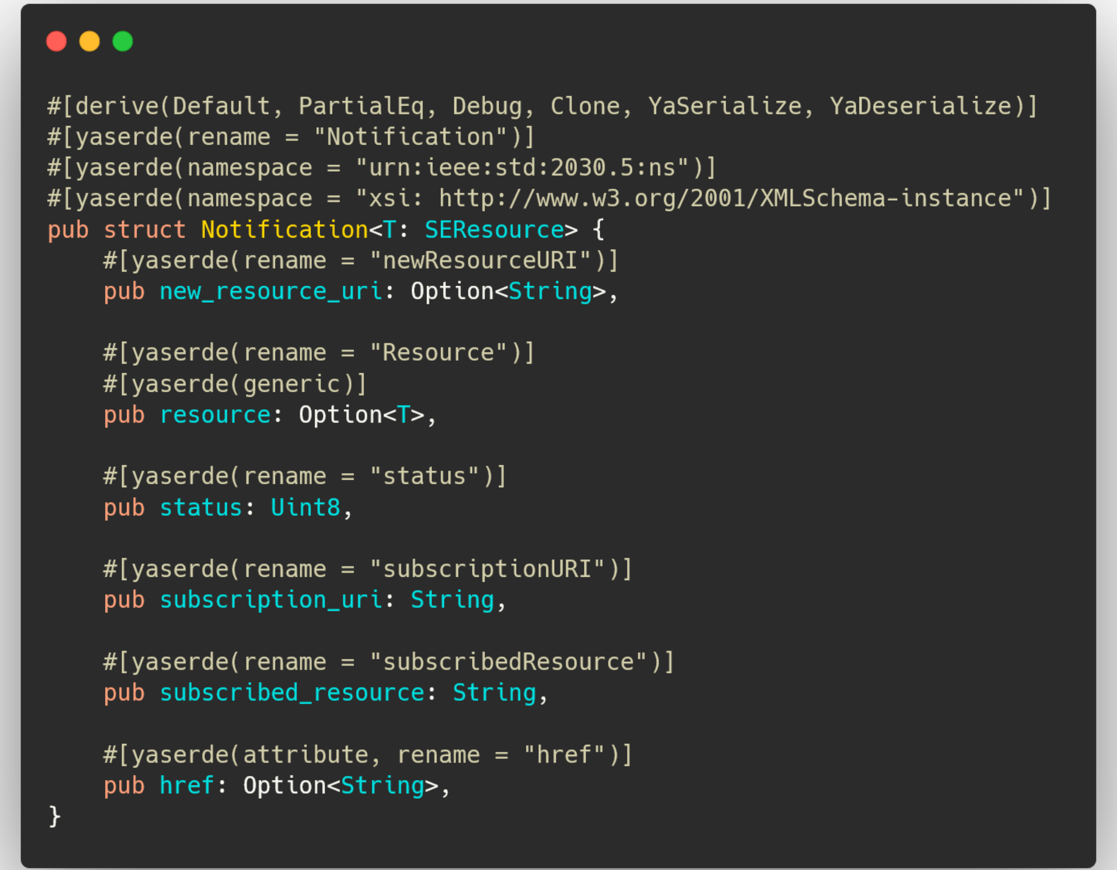
Progress:

Yaserde Changes

- yaserde_derive` - Procedural Macro
    - Modified Enum Serialisation / Deserialisation
- Generic Resources



```xml
<Notification xmlns="urn:ieee:std:2030.5:ns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Resource xsi:type="Reading">
    <timePeriod>
      <duration>0</duration>
      <start>12987364</start>
    </timePeriod>
    <value>1001</value>
  </Resource>
  <status>0</status>
  <subscriptionURI>/dev/8/sub/5</subscriptionURI>
  <subscribedResource>/upt/0/mr/4/r</subscribedResource>
</Notification>
```

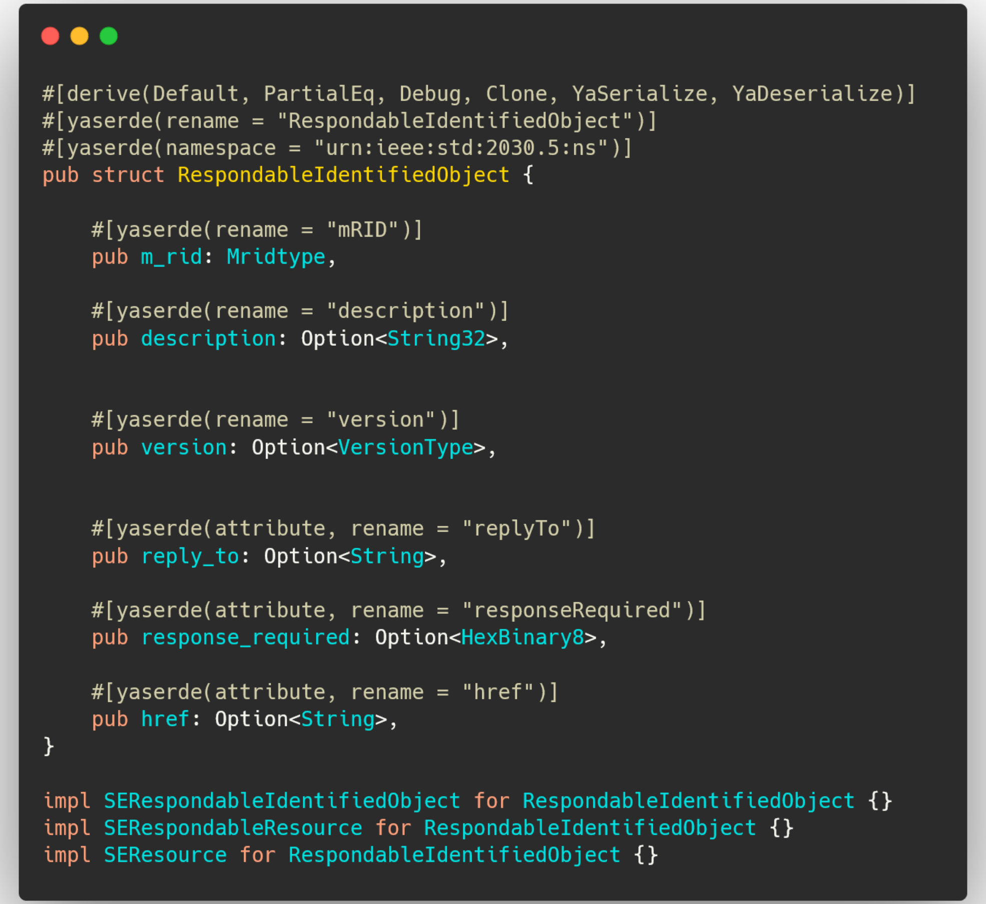


```rust
#[derive(Default, PartialEq, Debug, Clone, YaSerialize, YaDeserialize)]
#[yaserde(rename = "Notification")]
#[yaserde(namespace = "urn:ieee:std:2030.5:ns")]
#[yaserde(namespace = "xsi: http://www.w3.org/2001/XMLSchema-instance")]
pub struct Notification<T: SEResource> {
    #[yaserde(rename = "newResourceURI")]
    pub new_resource_uri: Option<String>,

    #[yaserde(rename = "Resource")]
    #[yaserde(generic)]
    pub resource: Option<T>,

    #[yaserde(rename = "status")]
    pub status: Uint8,

    #[yaserde(rename = "subscriptionURI")]
    pub subscription_uri: String,

    #[yaserde(rename = "subscribedResource")]
    pub subscribed_resource: String,

    #[yaserde(attribute, rename = "href")]
    pub href: Option<String>,
}
```

Progress:

xsd-parser-rs Changes

- Added Option<T> where applicable.
- Added empty trait implementations based off inheritance
- Added Yaserde renames to correct XML names
- Made all types derive 'Clone'.



```rust
#[derive(Default, PartialEq, Debug, Clone, YaSerialize, YaDeserialize)]
#[yaserde(rename = "RespondableIdentifiedObject")]
#[yaserde(namespace = "urn:ieee:std:2030.5:ns")]
pub struct RespondableIdentifiedObject {

    #[yaserde(rename = "mRID")]
    pub m_rid: Mridtype,

    #[yaserde(rename = "description")]
    pub description: Option<String32>,


    #[yaserde(rename = "version")]
    pub version: Option<VersionType>,


    #[yaserde(attribute, rename = "replyTo")]
    pub reply_to: Option<String>,

    #[yaserde(attribute, rename = "responseRequired")]
    pub response_required: Option<HexBinary8>,

    #[yaserde(attribute, rename = "href")]
    pub href: Option<String>,
}

impl SERespondableIdentifiedObject for RespondableIdentifiedObject {}
impl SERespondableResource for RespondableIdentifiedObject {}
impl SEResource for RespondableIdentifiedObject {}
```

Progress:

Dynamic Deserialisation

- Not particularly useful for our client.
- YaSerde needs to produce pointers to heap allocated resources
    - Implemented on a feature branch (Not used)
- Hypothetically:
    - Generated HashMap of type names to deserialization functions
    - Store using `dyn Trait`, Rust runtime polymorphsm.

Demonstration

- Client Binary, Server Binary
- Subscription / Notification implementation
- Unit Test Suite
- System Tests
- Q&A

Timeline

Today – Start of Term 3 (Thesis C)

- Redesigned Subscription / Notification Mechanism + Tests
- Resource Polling Service + Tests
- Distributed Energy Resources Tests
  - Ensuring we've implemented all functionality used in the DER client tests by EPRI.
- Finish Incomplete Function Sets
  - Implement LFDI & SFDI functions for Security FS.
  - Implement Metering FS functions.
- Start on Thesis C Timeline!

Timeline

Thesis C

- Implement List Invariants
- SEP Events / Event Queue
- Remaining FS Functionality.
  - Demand Response & Load Control
- Unit & System Tests
- Stretch Goals:
  - DNS-SD Server Discovery
  - Australian Specific CSIP Extensions
  - EXI Library for Rust
  - Contributing to rustTLS (Cipher suite impl.)