Documentation

Installation

To install and use the lib, you have two options: building from source or using the pre-built package from Maven Github Repository.

Build from Source

To build the project from source, follow these steps:

1. Clone the repository:

   ```
   git clone https://github.com/eclipse-tractusx/SSI-agent-lib.git
   ```

2. Navigate to the project directory:

   ```
   cd cx-ssi-lib
   ```

3. Build the project using Maven:

   ```
   mvn clean install
   ```

4. After a successful build, you can include the generated JAR file in your project's dependencies.

Use Maven Dependency

Alternatively, you can use the pre-built package available on Maven Central Repository by adding the following Maven dependency to your project's `pom.xml` file:

```
<dependency>
  <groupId>org.eclipse.tractusx.ssi</groupId>
  <artifactId>cx-ssi-agent-lib</artifactId>
  <version>0.0.3</version>
</dependency>
```

Make sure to update the version number if a newer version is available.

Once you've added the dependency, your build tool (e.g., Maven or Gradle) will automatically download the library and include it in your project.

Usage

To integrate this library into your SSI agent, follow these guidelines:

1. Import the required classes then Initialize SSiLibrary:

```
import org.eclipse.tractusx.ssi.lib.SsiLibrary;
public static void main(String[] args){
    SsiLibrary.initialize();
```

```
}
// ...
```

2. To bulid a DID Document:

```
import java.net.URI;
import java.util.ArrayList;
import java.util.List;

import org.eclipse.tractusx.ssi.lib.base.MultibaseFactory;
import org.eclipse.tractusx.ssi.lib.crypt.ed25519.Ed25519KeySet;
import org.eclipse.tractusx.ssi.lib.did.web.DidWebFactory;
import org.eclipse.tractusx.ssi.lib.model.MultibaseString;
import org.eclipse.tractusx.ssi.lib.model.did.Did;
import org.eclipse.tractusx.ssi.lib.model.did.VerificationMethod;


import org.eclipse.tractusx.ssi.lib.model.did.DidDocument;
import org.eclipse.tractusx.ssi.lib.model.did.DidDocumentBuilder;
import org.eclipse.tractusx.ssi.lib.model.did.Ed25519VerificationKey2020;
import org.eclipse.tractusx.ssi.lib.model.did.Ed25519VerificationKey2020Builder;
public static DidDocument buildDidDocument(String hostName,byte[] privateKey,byte[] publicKe
    final Did did = DidWebFactory.fromHostname(hostName);

    //Extracting keys
    final Ed25519KeySet keySet = new Ed25519KeySet(privateKey, publicKey);
    final MultibaseString publicKeyBase = MultibaseFactory.create(keySet.getPublicKey());



    //Building Verification Methods:
    final List<VerificationMethod> verificationMethods = new ArrayList<>();
    final Ed25519VerificationKey2020Builder builder = new Ed25519VerificationKey2020Builder(
    final Ed25519VerificationKey2020 key =
         builder
             .id(URI.create(did.toUri() + "#key-" + 1))
             .controller(did.toUri())
             .publicKeyMultiBase(publicKeyBase)
             .build();
    verificationMethods.add(key);

    final DidDocumentBuilder didDocumentBuilder = new DidDocumentBuilder();
    didDocumentBuilder.id(did.toUri());
    didDocumentBuilder.verificationMethods(verificationMethods);

    return didDocumentBuilder.build();
}
```

```
// ...
```

3. To Resolve DID document using DID Web:

```
import java.net.http.HttpClient;

import org.eclipse.tractusx.ssi.lib.did.web.DidWebDocumentResolver;
import org.eclipse.tractusx.ssi.lib.did.web.DidWebFactory;
import org.eclipse.tractusx.ssi.lib.did.web.util.DidWebParser;
import org.eclipse.tractusx.ssi.lib.exception.DidDocumentResolverNotRegisteredException;
import org.eclipse.tractusx.ssi.lib.model.did.Did;
import org.eclipse.tractusx.ssi.lib.model.did.DidDocument;
import org.eclipse.tractusx.ssi.lib.model.did.DidMethod;
import org.eclipse.tractusx.ssi.lib.resolver.DidDocumentResolverRegistryImpl;

public static DidDocument resovleDocument(String didUrl) throws DidDocumentResolverNotRegist

    //DID Resolver Constracture params
    DidWebParser didParser = new DidWebParser();
    var httpClient = HttpClient.newHttpClient();
    var enforceHttps = false;

    //DID Method
    DidMethod didWeb = new DidMethod("web");

    //DID
    Did did = DidWebFactory.fromHostname(didUrl);

    var didDocumentResolverRegistry = new DidDocumentResolverRegistryImpl();
    didDocumentResolverRegistry.register(new DidWebDocumentResolver(httpClient,didParser , e
    return didDocumentResolverRegistry.get(didWeb).resolve(did);
}
```

4. To Generate VerifiableCredential:

```
import java.net.URI;
import java.time.Instant;
import java.util.List;
import java.util.Map;

import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredential;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialBuilder;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialSubject;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialType;

public static VerifiableCredential createVCWithoutProof() {
```

```
    //VC Bulider
    final VerifiableCredentialBuilder verifiableCredentialBuilder =
    new VerifiableCredentialBuilder();

    //VC Subject
    final VerifiableCredentialSubject verifiableCredentialSubject =
    new VerifiableCredentialSubject(Map.of("test", "test"));

    //Using Builder
    final VerifiableCredential credentialWithoutProof =
    verifiableCredentialBuilder
        .id(URI.create("did:test:id"))
        .type(List.of(VerifiableCredentialType.VERIFIABLE_CREDENTIAL))
        .issuer(URI.create("did:test:isser"))
        .expirationDate(Instant.now().plusSeconds(3600))
        .issuanceDate(Instant.now())
        .credentialSubject(verifiableCredentialSubject)
        .build();

    return credentialWithoutProof;

}
```

5. To Generate VerifiableCredential with ED21559/JWS proof:

```
import java.net.URI;
import java.time.Instant;
import java.util.List;
import java.util.Map;

import org.eclipse.tractusx.ssi.lib.model.Ed25519Signature2020;
import org.eclipse.tractusx.ssi.lib.model.did.Did;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredential;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialBuilder;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialSubject;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredentialType;
import org.eclipse.tractusx.ssi.lib.proof.LinkedDataProofGenerator;


 public static VerifiableCredential createVCWithED21559Proof(
      VerifiableCredential credential, byte[] privateKey, Did issuer) {

    // VC Builder
    final VerifiableCredentialBuilder builder =
        new VerifiableCredentialBuilder()
```

```
            .context(credential.getContext())
            .id(credential.getId())
            .issuer(issuer.toUri())
            .issuanceDate(Instant.now())
            .credentialSubject(credential.getCredentialSubject())
            .expirationDate(credential.getExpirationDate())
            .type(credential.getTypes());

    // Ed25519 Proof Builder
    final LinkedDataProofGenerator generator = LinkedDataProofGenerator.newInstance(Signatur
    final Ed25519Signature2020 proof =
        (Ed25519Signature2020) generator.createProof(
        builder.build(), URI.create(issuer + "#key-1"), privateKey);

    // Adding Proof to VC
    builder.proof(proof);

    return builder.build();
  }

  public static VerifiableCredential createVCWithJWSProof(
    VerifiableCredential credential, byte[] privateKey, Did issuer) {

  // VC Builder
  final VerifiableCredentialBuilder builder =
      new VerifiableCredentialBuilder()
          .context(credential.getContext())
          .id(credential.getId())
          .issuer(issuer.toUri())
          .issuanceDate(Instant.now())
          .credentialSubject(credential.getCredentialSubject())
          .expirationDate(credential.getExpirationDate())
          .type(credential.getTypes());

  // JWS Proof Builder
  final LinkedDataProofGenerator generator = LinkedDataProofGenerator.newInstance(SignatureT
  final JWSSignature2020 proof =
      (JWSSignature2020) generator.createProof(
      builder.build(), URI.create(issuer + "#key-1"), privateKey);

  // Adding Proof to VC
  builder.proof(proof);

  return builder.build();
}
```

6. To Generate Verifiable Presentation:

```
import java.util.List;
import org.eclipse.tractusx.ssi.lib.model.did.Did;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredential;

public static VerifiablePresentation createVP( Did issuer, List<VerifiableCredential> creden
    //VP Builder
    final VerifiablePresentationBuilder verifiablePresentationBuilder =
        new VerifiablePresentationBuilder();

    // Build VP
    final VerifiablePresentation verifiablePresentation =
        verifiablePresentationBuilder
            .id(issuer.toUri()) // NOTE: Provide unique ID number to each VP you create!!
            .type(List.of(VerifiablePresentationType.VERIFIABLE_PRESENTATION))
            .verifiableCredentials(credentials)
            .build();
    return verifiablePresentation;
 }
```

7. To Generate Signed Verifiable Presentation:

```
import java.util.List;

import org.eclipse.tractusx.ssi.lib.crypt.ed25519.Ed25519Key;
import org.eclipse.tractusx.ssi.lib.crypt.ed25519.Ed25519KeySet;
import org.eclipse.tractusx.ssi.lib.jwt.SignedJwtFactory;
import org.eclipse.tractusx.ssi.lib.model.did.Did;
import org.eclipse.tractusx.ssi.lib.model.verifiable.presentation.VerifiablePresentation;
import org.eclipse.tractusx.ssi.lib.model.verifiable.presentation.VerifiablePresentationBuil
import org.eclipse.tractusx.ssi.lib.model.verifiable.presentation.VerifiablePresentationType
import org.eclipse.tractusx.ssi.lib.resolver.OctetKeyPairFactory;
import org.eclipse.tractusx.ssi.lib.serialization.jsonLd.JsonLdSerializerImpl;
import org.eclipse.tractusx.ssi.lib.serialization.jwt.SerializedJwtPresentationFactory;
import org.eclipse.tractusx.ssi.lib.serialization.jwt.SerializedJwtPresentationFactoryImpl;

import com.nimbusds.jwt.SignedJWT;


 public static SignedJWT createVPAsJWT(Did issuer,List<VerifiableCredential> credentials, St

    //Extracting keys
    final Ed25519KeySet keySet = new Ed25519KeySet(privateKey, publicKey);
    final Ed25519Key signingKey = new Ed25519Key(keySet.getPrivateKey());

    //JWT Factory
```

```
    final SerializedJwtPresentationFactory presentationFactory = new SerializedJwtPresentati
            new SignedJwtFactory(new OctetKeyPairFactory()), new JsonLdSerializerImpl(), iss

    //Build JWT
    return presentationFactory.createPresentation(
        issuer, credentials, audience, signingKey);


}
```

8. To Verfiy JWT (VC or VP) :

```
package org.eclipse.tractusx.ssi.examples;

import java.net.http.HttpClient;

import org.eclipse.tractusx.ssi.lib.did.web.DidWebDocumentResolver;
import org.eclipse.tractusx.ssi.lib.did.web.util.DidWebParser;
import org.eclipse.tractusx.ssi.lib.exception.DidDocumentResolverNotRegisteredException;
import org.eclipse.tractusx.ssi.lib.exception.JwtException;
import org.eclipse.tractusx.ssi.lib.jwt.SignedJwtVerifier;
import org.eclipse.tractusx.ssi.lib.resolver.DidDocumentResolverRegistryImpl;

import com.nimbusds.jwt.SignedJWT;

public static boolean verifyJWT(SignedJWT jwt) {
    // DID Resolver Constracture params
    DidWebParser didParser = new DidWebParser();
    var httpClient = HttpClient.newHttpClient();
    var enforceHttps = false;

    var didDocumentResolverRegistry = new DidDocumentResolverRegistryImpl();
    didDocumentResolverRegistry.register(
            new DidWebDocumentResolver(httpClient, didParser, enforceHttps));

    SignedJwtVerifier jwtVerifier = new SignedJwtVerifier(didDocumentResolverRegistry);
    try {
        return jwtVerifier.verify(jwt);
    } catch (JwtException | DidDocumentResolverNotRegisteredException e) {
        // An exception will be thrown here in case JWT verification failed or DID
        // Document Resolver not able to resolver.
        e.printStackTrace();
    }

}
```

9. To verify Json-LD:

```
import com.nimbusds.jwt.SignedJWT;
import java.net.http.HttpClient;
import org.eclipse.tractusx.ssi.lib.did.web.DidWebDocumentResolver;
import org.eclipse.tractusx.ssi.lib.did.web.util.DidWebParser;
import org.eclipse.tractusx.ssi.lib.model.verifiable.credential.VerifiableCredential;
import org.eclipse.tractusx.ssi.lib.proof.LinkedDataProofValidation;
import org.eclipse.tractusx.ssi.lib.resolver.DidDocumentResolverRegistryImpl;
import org.eclipse.tractusx.ssi.lib.model.proof.jws.JWSSignature2020;

 public static boolean verifyED21559LD(VerifiableCredential verifiableCredential) {
    // DID Resolver Constracture params
    DidWebParser didParser = new DidWebParser();
    var httpClient = HttpClient.newHttpClient();
    var enforceHttps = false;

    var didDocumentResolverRegistry = new DidDocumentResolverRegistryImpl();
    didDocumentResolverRegistry.register(
        new DidWebDocumentResolver(httpClient, didParser, enforceHttps));

    LinkedDataProofValidation proofValidation =
        LinkedDataProofValidation.newInstance(SignatureType.ED21559,didDocumentResolverRegis
    return proofValidation.verify(verifiableCredential);
  }

  public static boolean verifyJWSLD(VerifiableCredential verifiableCredential) {
    // DID Resolver Constracture params
    DidWebParser didParser = new DidWebParser();
    var httpClient = HttpClient.newHttpClient();
    var enforceHttps = false;

    var didDocumentResolverRegistry = new DidDocumentResolverRegistryImpl();
    didDocumentResolverRegistry.register(
        new DidWebDocumentResolver(httpClient, didParser, enforceHttps));

    LinkedDataProofValidation proofValidation =
        LinkedDataProofValidation.newInstance(SignatureType.JWS,didDocumentResolverRegistry)
    return proofValidation.verify(verifiableCredential);
  }
```

10. To Validate JWT expiry date and audience:

```
import com.nimbusds.jwt.SignedJWT;
import org.eclipse.tractusx.ssi.lib.exception.JwtAudienceCheckFailedException;
import org.eclipse.tractusx.ssi.lib.exception.JwtExpiredException;
import org.eclipse.tractusx.ssi.lib.jwt.SignedJwtValidator;
```

```
public class Validation {
    public static void validateJWTDate(SignedJWT signedJWT, String audience)
            throws JwtAudienceCheckFailedException, JwtExpiredException {
        SignedJwtValidator jwtValidator = new SignedJwtValidator();
        jwtValidator.validateDate(signedJWT);
    }

    public static void validateJWTAudiences(SignedJWT signedJWT, String audience)
            throws JwtAudienceCheckFailedException, JwtExpiredException {
        SignedJwtValidator jwtValidator = new SignedJwtValidator();
        jwtValidator.validateAudiences(signedJWT, audience);
    }
}
```

Architecture

Documentation Template: arc42

arc42, the template for documentation of software and system architecture. Template Version 8.2 EN. (based upon AsciiDoc version), January 2023 See arc42.org.

Introduction and Goals

The *SSI Agent Lib* (hereafter referred to as the **lib**) is an open-source Java library developed under the Tractus-X project. It provides core functionalities and abstractions commonly required when implementing a digital wallet or any service leveraging self-sovereign identities (SSI).

Requirements Overview

The lib supports the following use cases and interactions:

Feature	Description / Constraints
Create DID	
Parse DID	
Generate DID document	
Resolve DID document	
Issue Verifiable Credential	
Issue Verifiable Presentation	
Verify Verifiable Presentation	
Validate Verifiable Presentation	
Generate a key pair	Only Ed25519 supported.

Quality Goals

Priority	Quality Goal	Scenario
1	Flexibility	Support for multiple cryptographic algorithms.
1	Extensibility	Integration of custom implementations for certain aspects (e.g., DID resolution).
2	Usability	Seamless integration and usage within other systems.

Architecture Constraints

- Java is the designated programming language to ensure compatibility with the Managed Identity Wallet and the Tractus-X EDC.

- JWT based verifiable presentations are required for interoperability with the DAPS, which uses JWT Access-Tokens for AuthN/AuthZ.
- JsonWebKey2020 serves as the Crypto Suite for Verifiable Credentials (VCs) & Verifiable Presentations (VPs).

System Scope and Context

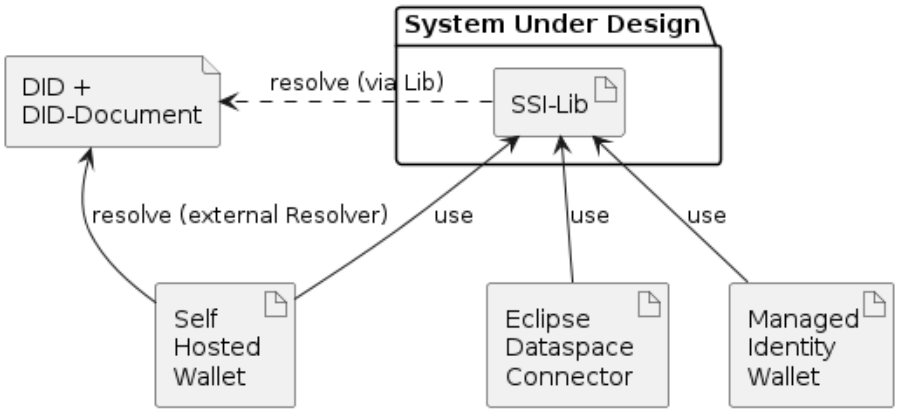

Figure 1: System Scope

The SSI Lib is intended for use by the Catena-X Managed Identity Wallet (MIW), the Eclipse Dataspace Connector (EDC), and third-party self-hosted wallets. While the SSI Lib provides did:web DID resolution capabilities, it also supports external DID resolution (e.g., Uniresolver).

Solution Strategy

The library adopts a stateless design with no data persistency. It offers segregated interfaces, allowing usage of both internal features and external components. For instance, internal and external

DID resolution can be swapped (see `DidDocumentResolver.java`).

Building Block View

Whitebox Overall System

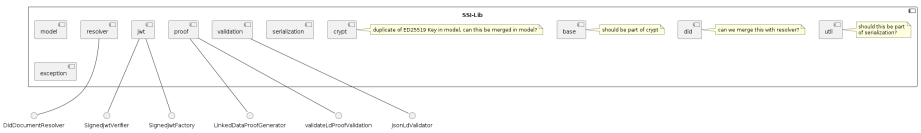

Figure 2: Whitebox System Overview

The library's building blocks are divided into various packages based on the provided SSI features, along with additional packages like `model` and `exception` for basic utilities.

Key Building Blocks

- `resolver`
- `jwt`
- `model`
- `proof`
- `serialization`
- `util`
- `validation`
- `exception`
- `did`
- `base`
- `crypt`

Crucial Interfaces

- `DidDocumentResolver`
- `LinkedDataProofGenerator`
- `validateLdProofValidator`
- `SignedJwtVerifier`
- `SignedJwtFactory`
- `JsonLdValidator`

resolver This component is responsible for resolving Decentralized Identifiers (DIDs). It interacts with the underlying infrastructure to retrieve and parse DID Documents associated with a given DID.

jwt The JWT (JSON Web Tokens) component is responsible for creating and verifying JWT-based verifiable presentations and credentials. It ensures the proper formatting and signing of JWTs.

model The model component contains the data structures and classes used across the library. It defines the main objects (like DID, Verifiable Credential, etc.) that the library operates on.

proof This component deals with the creation and validation of Linked Data Proofs. It generates proofs for Verifiable Credentials and validates incoming proofs.

serialization The serialization component converts between the library's internal data structures and the JSON-LD format used in SSI. It is essential for the import and export of SSI data.

util The util (or utility) component includes helper functions and classes used across the library. This may involve utility functions for encoding/decoding, date and time handling, etc.

validation The validation component verifies that data is correctly formatted and valid according to the defined schemas and specifications. It is used in multiple contexts, such as when receiving Verifiable Credentials or Verifiable Presentations.

exception The exception component defines the error and exception classes used in the library. It provides structured error handling and aids in debugging and error tracking.

did The DID component involves all functionality specifically related to DIDs, such as generation, parsing, and formatting of DIDs and DID Documents.

base The base component includes fundamental functionality and classes used throughout the library, setting the base structure of the library.

crypt The crypt (or cryptography) component is responsible for all cryptographic operations, like signing, verification, and key generation. It directly supports JsonWebKey2020 based operations.

Runtime View

Refer to the respective Feature Specs for insights into the library's runtime behavior.

Deployment View

The SSI Lib can be integrated into an application as a standard JAR file through common build tools (i.e., Maven, Gradle, etc.). Therefore, no additional deployment artifacts are necessary.

Cross-cutting Concepts

Extensibility

The architecture is designed to allow for the easy addition and integration of new features or alterations to existing ones. This is evident in the support for custom implementations (e.g., DID resolution) and the use of interfaces to allow flexibility in the underlying implementations.

Exception Handling

Exception handling is a recurring concept in the architecture. The library has the exception building block, and other building blocks should follow consistent practices for error/exception handling to ensure robust operation.

Architecture Decisions

Quality Requirements

- The library should create a JWT-based proof via JsonWebKey2020 / ED25519 signature within 0.5 seconds on current-generation server hardware under normal load (< 50% CPU Utilization)

Risks and Technical Debts

- Currently, only ED25519 is supported.
- No formal interface exists for key encoding; it currently uses a byte array.

Glossary

Term	Definition
EDC	Eclipse Dataspace Connector
MIW	Managed Identity Wallet
SSI	Self-Sovereign Identity

Feature: Create DID

1. Specification

Create a Decentralized Identifier (DID) as specified in W3C-DID-Core, for a set of supported DID methods.

Example:

```
did:web:mydomain.com:12345
```

1.1 Assumptions There is no need to ensure uniqueness of the created DID.

1.2 Constraints Currently only DID method **did:web** *MUST* be supported.

1.3 System Environment Any kind of registration process of a DID is out of scope and needs to be handled by the client.

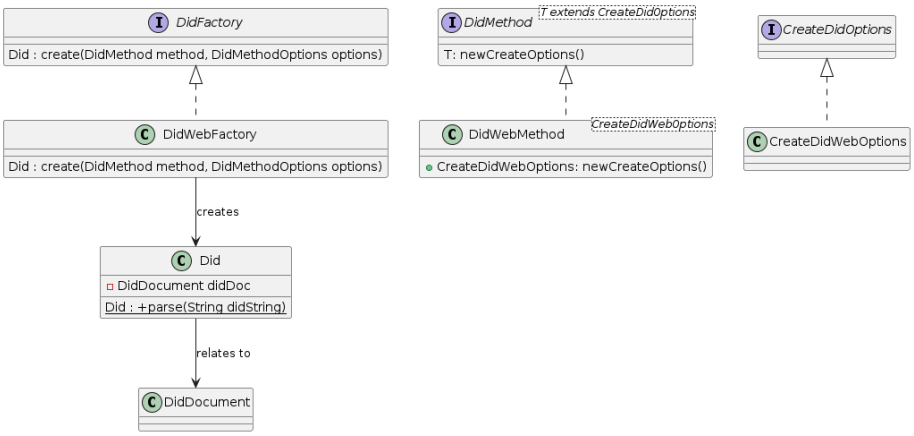

Figure 3: CreateParseDid.png

2. Architecture

2.1 Class Diagram

- DidFactory - Public factory interface.
- DidMethod - Defines a DID method, and allows retrieving a **CreateDidOptions** object specific to the respective DID method.
- CreateDidOptions - Marker interface. Implementations hold properties required to create a new DID of the respective **DidMethod**.
- DidFactoryRegistry - *MAY* be used to register **DidFactory** implementations for multiple **DidMethod**s
- DidWebMethod - Example implementation of **DidMethod** for method *did:web*.
- CreateDidWebOptions - Example implementation of **CreateDidOptions** for method *did:web*.
- Did - Value class representing a DID. *MAY* refer to a **DidDocument**
- DidDocument - Value class representing a DID document.

Feature: Generate DID Document

1. Specification

Given a valid DID, generate DID document as specified in W3C-DID-Core.

Example:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1",
```

```
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:web:mydomain.com:12345",
  "verificationMethod": [{
    "id": "did:web:mydomain.com:12345#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
    "type": "JsonWebKey2020",
    "controller": "did:web:mydomain.com:12345",
    "publicKeyJwk": {
      "crv": "Ed25519",
      "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ",
      "kty": "OKP",
      "kid": "_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A"
    }
  }, {
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }],
}
```


1.1 Assumptions Multiple verification methods *SHOULD* be supported.

1.2 Constraints Currently only verification type **Ed25519VerificationKey2020** needs to be supported.

2. Architecture

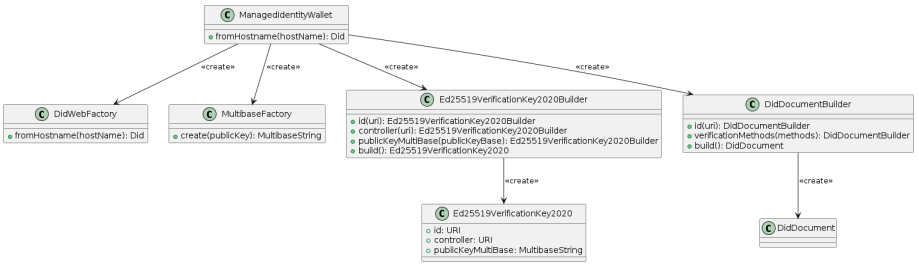

Figure 4: CreateDidClass.png

2.1 Class Diagrams

2.2 Sequence Diagrams *You can find an Example of the class interactions here:* /src/main/java/org/eclipse/tractusx/ssi/examples/BuildDIDDoc.java

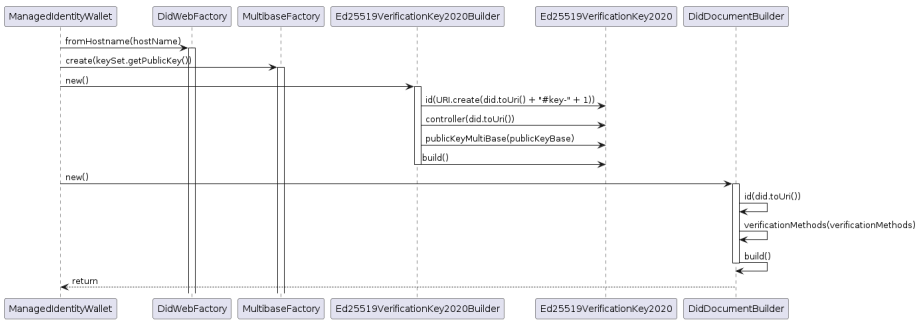

Figure 5: CreateDidSequence.png

Feature: Generate Key Pair

1. Specification

Given a supported key algorithm, generate a public / private key pair.

OPTIONAL: - Generated keys *MAY* be returned as strings, encoded in a supported encoding. - The seed used to initialize the random number generator *SHOULD* be returned. - A seed *MAY* be specified to allow generating pseudo-random key pair (e.g. for testing purposes).

Example:

```
{
    "type": "Ed25519VerificationKey2020",
    "publicKeyMultibase": "z6Mkqhx5Go6yU6yVt7vsWvu4QFPW5KMVGZmQASeiAdZ9ZmXL",
    "privateKeyMultibase": "zrv4DKJ9CLMzdmPanZmEi49nNMzj8MaHBH2CMfRQVdAr4FY1mpfex9qTGboUdmwv
}
```

1.1 Assumptions Multiple key algorithms *SHOULD* be supported.

1.2 Constraints Currently only verification type **Ed25519VerificationKey2020** needs to be supported.

2. Architecture

2.1 Class Diagrams

2.2 Sequence Diagrams

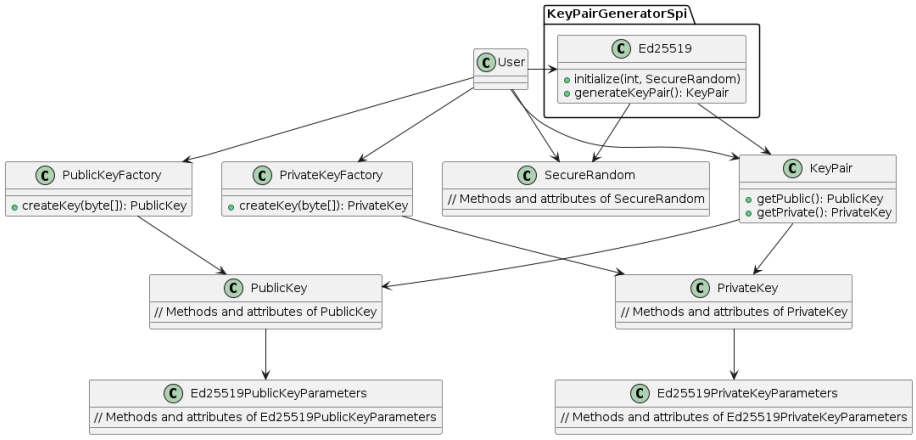

Figure 6: CreateKeypairEd25519Class.png

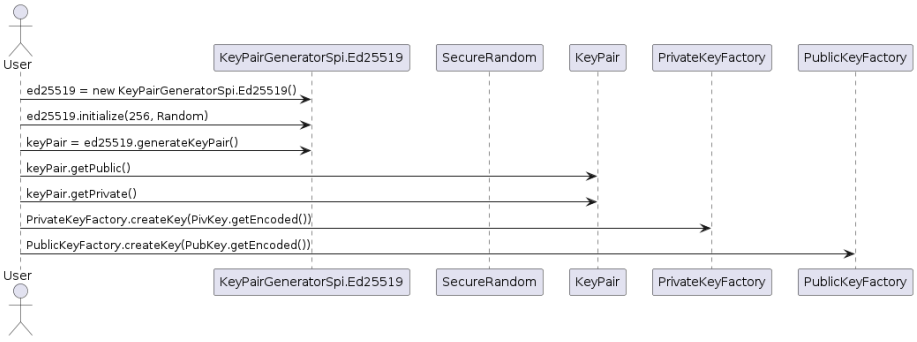

Figure 7: CreateKeypairEd25519Sequence.png

Feature: Issue Verifiable Presentation

1. Specification

Given a JSON-LD, an issuer DID and a supported signature algorithm, generate a proof as specified in the W3C VC-data-model, section 6.3.2 and return the signed verifiable credential.

Example:


```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "http://example.edu/credentials/1872",
  "type": ["VerifiableCredential", "AlumniCredential"],
  "issuer": "https://example.edu/issuers/565049",
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [{
        "value": "Example University",
        "lang": "en"
      }, {
        "value": "Exemple d'Université",
        "lang": "fr"
      }]
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-06-18T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.edu/issuers/565049#key-1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TCYt5X
      sITJX1CxPCT8yAV-TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq-pQy7UJiN5mgRxD-WUc
      X16dUEMGlv50aqzpqh4Qktb3rk-BuQy72IFLOqV0G_zS245-kronKb78cPN25DGlcTwLtj
      PAYuNzVBAh4vGHSrQyHUdBBPM"
  }
}
```


1.1 Assumptions Multiple signature algorithms *SHOULD* be supported.

1.2 Constraints Currently only verification type **Ed25519Signature2020** needs to be supported.

2. Architecture

2.1 Class Diagrams *Provide here any class diagrams needed to illustrate the application. These can be ordered by which component they construct or contribute to. If there is any ambiguity in the diagram or if any piece needs more description provide it here as well in a subsection.*

2.2 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

Feature: Issue Verifiable Presentation

1. Specification

Given a verifiable credential, an issuer DID and an audience, return a verifiable presentation as a JWT.

Example: JWT payload


```
{
  "iss": "did:web:localhost%3A8080",
  "sub": "did:web:localhost%3A8080",
  "aud": "test",
  "vp": {
    "id": "did:web:localhost%3A8080#fd10a61d-3726-45e7-8355-db5f3a4dbe60",
    "type": [
      "VerifiablePresentation"
    ],
    "@context": [
      "https://www.w3.org/2018/credentials/v1"
    ],
    "verifiableCredential": {
      "@context": [
        "https://www.w3.org/2018/credentials/v1"
      ],
      "type": [
        "VerifiableCredential"
      ],
      "id": "https://localhost:8080/12345",
      "issuer": "did:web:localhost%3A8080",
      "issuanceDate": "2023-05-26T13:58:00Z",
      "expirationDate": "2000-01-23T04:56:07Z",
      "credentialSubject": {
```

```
        "name": "Jane Doe",
        "id": "did:example:abcdef1234567"
      },
      "proof": {
        "proofPurpose": "proofPurpose",
        "type": "Ed25519Signature2020",
        "proofValue": "zLLs4YXK4dhsaifJGmeyp23TsyUnGxJkobsT8fDgzXdq27dKFSgbXwvb857VyXRtBSLv2
        "verificationMethod": "did:web:localhost%3A8080#key-1",
        "created": "2023-05-26T13:58:00Z"
      }
    }
  },
  "exp": 1686311279,
  "jti": "89c16630-69ca-4b18-baa7-e93d3d3a016c"
}
```


1.1 Assumptions Multiple signature algorithms *SHOULD* be supported.

1.2 Constraints Currently only verification type **Ed25519Signature2020** needs to be supported.

2. Architecture

2.1 Class Diagrams *Provide here any class diagrams needed to illustrate the application. These can be ordered by which component they construct or contribute to. If there is any ambiguity in the diagram or if any piece needs more description provide it here as well in a subsection.*

2.2 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

Feature: Parse DID

1. Specification

Create a Decentralized Identifier (DID) as specified in W3C-DID-Core, for a set of supported DID methods.

Example:

`did:web:mydomain.com:12345`

1.1 Assumptions There is no need to ensure uniqueness of the created DID.

1.2 Constraints Currently only DID method **did:web** *MUST* be supported.

1.3 System Environment Any kind of registration process of a DID is out of scope and needs to be handled by the client.

2. Architecture

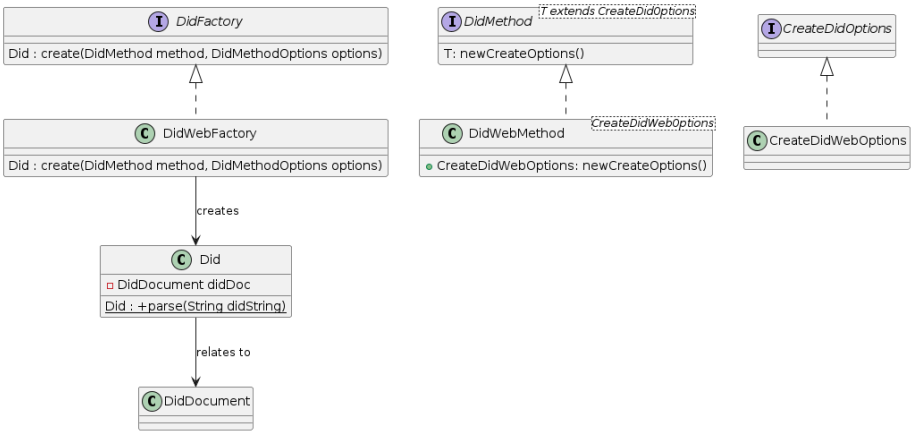

Figure 8: ResolveDIDdoc.png

2.1 Class Diagrams

2.2 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

Feature: Resolve DID Document

1. Specification

Given a valid DID, retrieve the respective DID document as specified in W3C-DID-Core, for a set of supported DID methods.

Example:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:web:mydomain.com:12345",
  "verificationMethod": [{
    "id": "did:web:mydomain.com:12345#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
    "type": "JsonWebKey2020",
```

```
    "controller": "did:web:mydomain.com:12345",
    "publicKeyJwk": {
      "crv": "Ed25519",
      "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ",
      "kty": "OKP",
      "kid": "_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A"
    }
  }, {
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:pqrstuvwxyz0987654321",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }],
}
```


1.1 Assumptions

- There *MAY* be muliple resolvers available for a given DID method.
- The priority of the resolvers *MUST* be customizable.
- A resolver *MAY* support multiple DID methods.

1.2 Constraints *none*

1.3 System Environment If the resolver is running as a separate process, all operational & communication aspects are out of scope.

2. Architecture

2.1 Overview Define a public resolver interface and exception class. This enables clients to freely choose the provided implementations or use a custom one. The *isResolvable* method *SHOULD* be used to determine whether the resolver is able to resolve the DID document of a provided DID without actually doing it, which allows to apply a ressource efficient 'fail early' strategy. Support for multiple resolvers is achieved by a resolver that applies the Composite Pattern to execute the provided resolvers in sequence.

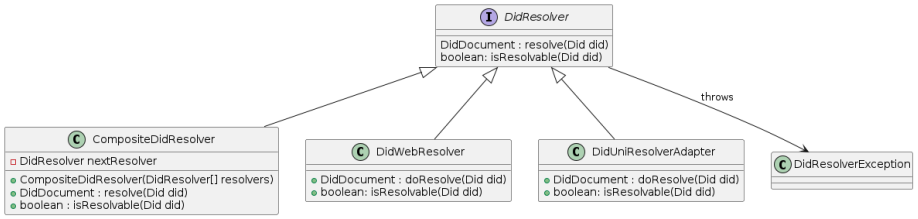

Figure 9: ResolveDidDoc.png

2.2 Class Diagrams

- DidResolver - Public interface to be used by clients.
- DidResolverException - Exception class to be thrown when a DID cannot be resolved.
- DidWebResolver / DidUniResolverAdapter - Examples of implementations of the *DidResolver* interface.
- CompositeDidResolver - *DidResolver* implementation that is able to chain multiple resolvers. It may execute the *resolve* method of each provided resolver until a DID document is returned.

2.3 Sequence Diagrams The following diagram illustrates how multiple *DidResolver* implementations can be orchestrated to a chain that is capable of resolving DID of different types or anchored in different registries (VDRs):

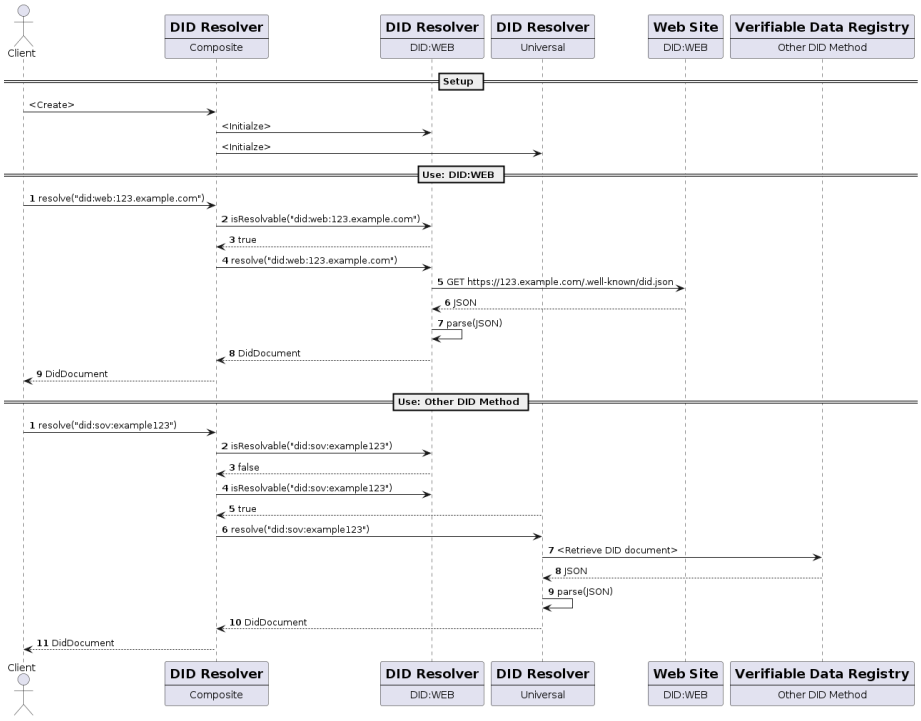

Figure 10: DidResolutionFlow.png

Feature: Validate Verifiable Presentation

1. Specification

Given a verifiable presentation, evaluate if the presention is valid. The validity *MAY* be determined based on different criteria, e.g. - expiration date - expected audience

Example: JWT payload


```
{
  "iss": "did:web:localhost%3A8080",
  "sub": "did:web:localhost%3A8080",
  "aud": "test",
  "vp": {
    "id": "did:web:localhost%3A8080#fd10a61d-3726-45e7-8355-db5f3a4dbe60",
    "type": [
      "VerifiablePresentation"
    ],
    "@context": [
      "https://www.w3.org/2018/credentials/v1"
    ],
    "verifiableCredential": {
      "@context": [
        "https://www.w3.org/2018/credentials/v1"
      ],
      "type": [
        "VerifiableCredential"
      ],
      "id": "https://localhost:8080/12345",
      "issuer": "did:web:localhost%3A8080",
      "issuanceDate": "2023-05-26T13:58:00Z",
      "expirationDate": "2000-01-23T04:56:07Z",
      "credentialSubject": {
        "name": "Jane Doe",
        "id": "did:example:abcdef1234567"
      },
      "proof": {
        "proofPurpose": "proofPurpose",
        "type": "Ed25519Signature2020",
        "proofValue": "zLLs4YXK4dhsaifJGmeyp23TsyUnGxJkobsT8fDgzXdq27dKFSgbXwvb857VyXRtBSLv2
        "verificationMethod": "did:web:localhost%3A8080#key-1",
        "created": "2023-05-26T13:58:00Z"
      }
    }
  },
  "exp": 1686311279,
```

```
  "jti": "89c16630-69ca-4b18-baa7-e93d3d3a016c"
}
```

1.1 Assumptions Multiple signature algorithms *SHOULD* be supported.

1.2 Constraints Currently only verification type **Ed25519Signature2020** needs to be supported.

2. Architecture

2.1 Class Diagrams *Provide here any class diagrams needed to illustrate the application. These can be ordered by which component they construct or contribute to. If there is any ambiguity in the diagram or if any piece needs more description provide it here as well in a subsection.*

2.2 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

Feature: Verify Verifiable Presentation

1. Specification

Given a verifiable presentation, verify if the provided signatures are valid.

1.1 Assumptions Multiple signature algorithms *SHOULD* be supported.

1.2 Constraints Currently only verification types **Ed25519Signature2020** and **JsonWebKey2020** need to be supported.

2. Architecture

2.1 Class Diagrams *Provide here any class diagrams needed to illustrate the application. These can be ordered by which component they construct or contribute to. If there is any ambiguity in the diagram or if any piece needs more description provide it here as well in a subsection.*

2.2 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

Testing: Create DID

1. Specification

Create a Decentralized Identifier (DID) as specified in W3C-DID-Core, for a set of supported DID methods.

Example:

`did:web:mydomain.com:12345`

1.1 Assumptions There is no need to ensure uniqueness of the created DID.

1.2 Constraints Currently only DID method **did:web** *MUST* be supported.

1.3 System Environment Any kind of registration process of a DID is out of scope and needs to be handled by the client.

2. Architecture

2.1 Overview *Provide here a descriptive overview of the software/system/application architecture.*

2.2 Component Diagrams *Provide here the diagram and a detailed description of its most valuable parts. There may be multiple diagrams. Include a description for each diagram. Subsections can be used to list components and their descriptions.*

2.3 Class Diagrams *Provide here any class diagrams needed to illustrate the application. These can be ordered by which component they construct or contribute to. If there is any ambiguity in the diagram or if any piece needs more description provide it here as well in a subsection.*

2.4 Sequence Diagrams *Provide here any sequence diagrams. If possible list the use case they contribute to or solve. Provide descriptions if possible.*

2.5 Deployment Diagrams *Provide here the deployment diagram for the system including any information needed to describe it. Also, include any information needed to describe future scaling of the system.*

2.6 Other Diagrams *Provide here any additional diagrams and their descriptions in subsections.*

3 User Interface Design

Provide here any user interface mock-ups or templates. Include explanations to describe the screen flow or progression.

4 Appendices and References

4.1 Definitions and Abbreviations *List here any definitions or abbreviations that could be used to help a new team member understand any jargon that is frequently referenced in the design document.*

4.2 References *List here any references that can be used to give extra information on a topic found in the design document. These references can be referred to using superscript in the rest of the document.*

NOTICE