# **Appendix 2 - Version 1.3 \_12/12/2023**

The following sections of code are taken from the source code that was amended so as to run the proof of concept. Each of the sections are delineated by a **~~~~~~** and each section of source code have interlaced screen shots from the code that was ran and committed to my GitHub repository.

# **types.ts**

**Organising types**

One of the primary purposes of a types.ts file is to define custom types. TypeScript allows developers to create their own types using interfaces, type aliases, and enums. These custom types can be used to describe the shape and structure of data, making the code more readable and maintainable.

The code snippet provided is importing various modules and classes from the @iota/identity-wasm/node and @iota/sdk-wasm/node packages, as well as custom abstractions for identity and presentation.

**//INSIDE OF THE SCRIPT**

import type {

Credential,

IotaDocument,

IotaIdentityClient,

Jwt,

JwtCredentialValidator,

JwtPresentationValidator,

Resolver,

Status,

Storage,

} from "@iota/identity-wasm/node";

import { AliasOutput, Client } from "@iota/sdk-wasm/node";

import { Identity } from "./abstraction/identity";

import { PresentationRequest } from "./abstraction/presentation";

Credential: This import represents the Credential type, which is used to define and manage credentials within the IOTA DID framework. Credentials are verifiable claims about an entity's attributes or qualifications.

IotaDocument**:** This import represents the IotaDocument type, which is used to define and manage the decentralized identity documents within the IOTA DID framework. A decentralized identity document contains information about an entity's identity, such as public keys and service endpoints.

IotaIdentityClient**:** This import represents the IotaIdentityClient class, which is used to interact with the IOTA Tangle network and perform operations related to decentralized identities. It provides methods for creating and updating identity documents, issuing and verifying credentials, and more.

Jwt**:** This import represents the Jwt type, which is used to handle JSON Web Tokens (JWTs) within the IOTA DID framework. JWTs are used for secure communication and authentication between different parties.

JwtCredentialValidator**:** This import represents the JwtCredentialValidator class, which is used to validate the authenticity and integrity of credentials issued by trusted issuers. It verifies the signature and other claims of a JWT credential.

JwtPresentationValidator**:** This import represents the JwtPresentationValidator class, which is used to validate the authenticity and integrity of presentations (proofs) provided by entities. It verifies the signature and other claims of a JWT presentation.

Resolver**:** This import represents the Resolver type, which is used to resolve and retrieve decentralized identity documents from the IOTA Tangle network. It provides methods for resolving DID URLs and fetching the corresponding identity documents.

Status**:** This import represents the Status type, which is used to represent the status of various operations within the IOTA DID framework. It can indicate success, failure, or other relevant statuses.

Storage**:** This import represents the Storage type, which is used to define the storage mechanism for identity documents and credentials. It provides methods for storing and retrieving data related to decentralized identities.

AliasOutput**:** This import represents the AliasOutput class from the @iota/sdk-wasm/node package. It is used for interacting with the IOTA Tangle network and performing operations related to transactions and outputs.

Client**:** This import represents the Client class from the @iota/sdk-wasm/node package. It is used for interacting with the IOTA Tangle network and performing various operations, such as sending transactions and fetching data.

Identity**:** This import represents a custom abstraction for identity, which is defined in the ./abstraction/identity file. It encapsulates the logic and functionality related to managing and interacting with decentralized identities.

PresentationRequest**:** This import represents a custom abstraction for presentation requests, which is defined in the ./abstraction/presentation file. It encapsulates the logic and functionality related to creating and handling presentation requests within the IOTA DID framework.

A computer screen with colorful text

Description automatically generated

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The IdentityState type in the provided code snippet represents the state of an identity within the IOTA DID framework. It consists of several properties that are essential for managing and interacting with the identity.

**//INSIDE OF THE SCRIPT**

export type IdentityState = {

client: IotaIdentityClient;

address: string;

storage: Storage;

document: IotaDocument;

mnemonic: string;

};

client: IotaIdentityClient: This property represents the IOTA Identity Client, which is responsible for interacting with the IOTA Tangle and performing operations related to the identity. The client provides methods for creating and updating DIDs (Decentralized Identifiers), as well as signing and verifying messages.

address: string: This property stores the address associated with the identity. In the IOTA Tangle, addresses are used to receive and send transactions. The address is typically derived from the DID and serves as a unique identifier for the identity.

storage: Storage: This property represents the storage mechanism used to persist the identity's data. It can be any implementation of the Storage interface, which defines methods for reading and writing data related to the identity. The storage can be a local database, a distributed ledger, or any other suitable storage solution.

document: IotaDocument: This property holds the IOTA Document associated with the identity. The IOTA Document is a JSON-LD document that contains information about the identity, such as public keys, authentication methods, and service endpoints. It serves as a self-contained representation of the identity's attributes and can be used for verification and authentication purposes.

mnemonic: string: This property stores the mnemonic phrase associated with the identity. A mnemonic phrase is a human-readable representation of a cryptographic seed, which is used to derive the private keys for signing transactions and messages. The mnemonic allows for easy backup and recovery of the identity's keys.

A black rectangle with white text

Description automatically generated

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The provided code snippet is written in Typescript and defines a type called RevocationState. This type is used to represent the revocation state of a decentralized identifier (DID) within the IOTA DID framework.

**//INSIDE OF THE SCRIPT**

export type RevocationState = {

issuer: Identity;

output?: AliasOutput;

};

export type RevocationState: This line declares a new type called RevocationState and exports it, making it accessible to other parts of the codebase. This type represents the revocation state of a DID.

{ issuer: Identity; output?: AliasOutput; }: This is the structure of the RevocationState type. It consists of two properties:

issuer: This property represents the identity (issuer) associated with the revocation state. The Identity type is expected to be defined elsewhere in the codebase.

output?: AliasOutput: This property represents the output of the revocation state, specifically an AliasOutput. The AliasOutput type is optional (? denotes optional), meaning it may or may not be present.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The provided Typescript code snippet defines a type called CredentialState within the IOTA DID framework.

**//INSIDE OF THE SCRIPT**

export type CredentialState = {

issuer: Identity;

subject?: Identity;

properties: { [key: string]: string };

verificationMethod?: string;

revocationStatus?: Status;

};

issuer: Identity: This property represents the identity of the entity that issued the credential. In the context of the IOTA DID framework, an identity is a unique identifier associated with a specific entity or organization.

subject?: Identity: This property represents the identity of the subject to whom the credential is issued. The subject property is optional, indicating that the credential may or may not be associated with a specific subject.

properties: { [key: string]: string }: This property represents a collection of key-value pairs that define the properties or attributes associated with the credential. The properties object allows for the inclusion of various information related to the credential, such as the type of credential, expiration date, or any other relevant data.

verificationMethod?: string: This property represents the verification method used to validate the authenticity and integrity of the credential. The verificationMethod is optional, indicating that the credential may or may not require a specific verification method.

revocationStatus?: Status: This property represents the revocation status of the credential. The revocationStatus is optional and can be used to indicate whether the credential has been revoked or is still valid.

A black screen with colorful text

Description automatically generated

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

In the IOTA DID framework, the Predicate type is used to define a function that takes a Credential object as input and returns a boolean value. This function is used to evaluate whether a given credential satisfies certain conditions or criteria.

export type Predicate = (credential: Credential) => boolean;

export type Predicate: This line declares a new type called Predicate and exports it, making it available for use in other parts of the codebase.

(credential: Credential) => boolean: This part of the definition specifies the function signature of the Predicate type. It indicates that the Predicate type is a function that takes a single parameter of type Credential and returns a boolean value.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The provided code snippet is written in TypeScript and defines the PresentationRequestOptions type. This type is used to specify the options for a presentation request in the context of the IOTA DID framework.

**//INSIDE OF THE SCRIPT**

export type PresentationRequestOptions = {

predicates?: Predicate[];

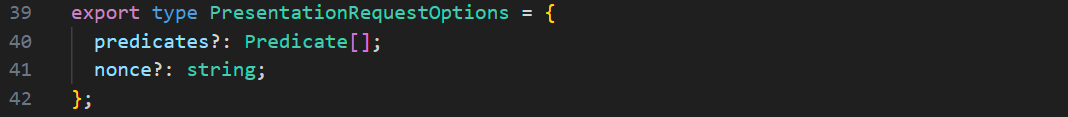
nonce?: string;

};

export type PresentationRequestOptions: This line declares a new type called PresentationRequestOptions and exports it. This type represents the options for a presentation request.

predicates?: Predicate[]: This line defines an optional property called predicates of type Predicate[]. The Predicate type represents a condition that needs to be satisfied for the presentation request. The predicates property allows specifying multiple predicates for the request.

nonce?: string: This line defines an optional property called nonce of type string. The nonce property is used to ensure the uniqueness of the presentation request. It is a random value generated by the requester and included in the request. The nonce helps prevent replay attacks.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The provided Typescript code snippet defines the PresentationResponseState type in the context of the IOTA DID framework.

**//INSIDE OF THE SCRIPT**

export type PresentationResponseState = {

holder: Identity;

nonce?: string;

verificationMethod?: string;

credentials: Jwt[];

};

holder: Identity: This property represents the identity of the holder of the presentation response. In the IOTA DID framework, an identity is a unique identifier associated with a specific entity or individual.

nonce?: string: The nonce property is an optional string that serves as a cryptographic nonce. A nonce is a random value used to ensure the uniqueness and integrity of cryptographic operations. In the context of the IOTA DID framework, the nonce can be used to prevent replay attacks or to add an additional layer of security to the presentation response.

verificationMethod?: string: The verificationMethod property is an optional string that specifies the method used for verifying the authenticity and integrity of the presentation response. In the IOTA DID framework, a verification method can be a public key or any other mechanism used to validate the digital signature associated with the presentation response.

credentials: Jwt[]: The credentials property is an array of JSON Web Tokens (JWTs) that represent the credentials associated with the presentation response. JWTs are a widely used standard for representing claims between two parties. In the context of the IOTA DID framework, these credentials can include verifiable claims or proofs that provide evidence of the holder's identity or attributes.

A screen shot of a computer

Description automatically generated

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The CreatePresentationValidationState type is a TypeScript interface that represents the state of a presentation validation process in the IOTA DID framework. It contains several properties that are used to store and manage the necessary information and objects for validating a presentation.

**//INSIDE OF THE SCRIPT**

export type CreatePresentationValidationState = {

request?: PresentationRequest;

response?: Jwt;

resolver: Resolver;

presentationValidator: JwtPresentationValidator;

credentialValidator: JwtCredentialValidator;

};

request?: PresentationRequest: This property represents the presentation request object. It is an optional property because not all validation processes may require a presentation request. The presentation request contains the details of the requested presentation, such as the required credentials or attributes.

response?: Jwt: This property represents the response object. It is also an optional property because not all validation processes may have a response object. The response object contains the presentation data that is being validated.

resolver: Resolver: This property represents the resolver object. The resolver is responsible for resolving and retrieving the necessary information from the decentralized identity network. It is used to fetch the required credentials or attributes for the presentation validation process.

presentationValidator: JwtPresentationValidator: This property represents the presentation validator object. The presentation validator is responsible for validating the presentation data against the presentation request. It verifies the authenticity and integrity of the presentation and ensures that it meets the required criteria.

credentialValidator: JwtCredentialValidator: This property represents the credential validator object. The credential validator is responsible for validating the credentials included in the presentation. It verifies the authenticity and integrity of the credentials and ensures that they are valid and issued by trusted sources.

A screen shot of a computer program

Description automatically generated

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The provided code snippet is written in TypeScript and defines a type called PresentationValidationOptions. This type is used to represent the options for validating a presentation in the context of the IOTA DID framework.

**//INSIDE OF THE SCRIPT**

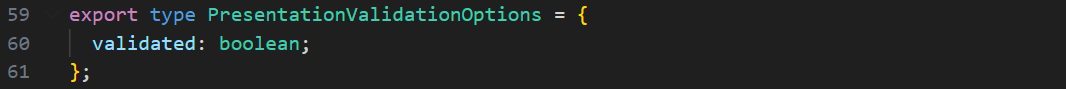
export type PresentationValidationOptions = {

validated: boolean;

};

In the code snippet, PresentationValidationOptions is defined as an object with a single property called validated, which is of type boolean. This property is used to indicate whether a presentation has been successfully validated or not.

The purpose of this type is to provide a structured way to pass validation options to functions or methods that deal with presentations in the IOTA DID framework.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

The CoverStorageDeposit function in the IOTA DID framework is a type definition that represents a function responsible for covering the storage deposit required for a specific address.

**//INSIDE OF THE SCRIPT**

export type CoverStorageDeposit = (

address: string,

tokensRequired: bigint

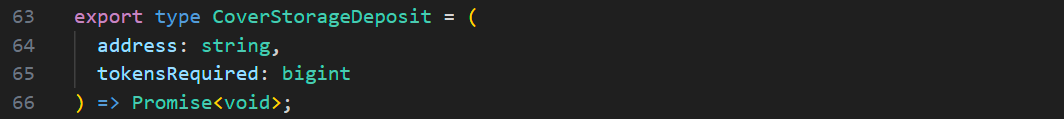
) => Promise<void>;

In this code snippet, we can see that CoverStorageDeposit is defined as a type. It represents a function that takes two parameters: address and tokensRequired.

The address parameter is of type string and represents the address for which the storage deposit needs to be covered.

The tokensRequired parameter is of type bigint and represents the number of tokens required to cover the storage deposit.

The function returns a Promise<void>, indicating that it is an asynchronous function that does not return any value.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**//INSIDE OF THE SCRIPT**

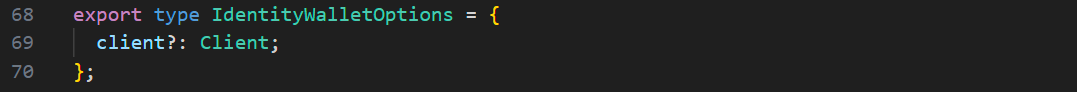
export type IdentityWalletOptions = {

client?: Client;

};

The IdentityWalletOptions type in the IOTA DID framework is used to define the options that can be passed when creating an identity wallet. In the provided code snippet, the IdentityWalletOptions type is being used to define an object with an optional property client of type Client.

The client property is used to specify the IOTA client that will be used by the identity wallet. The Client type represents the IOTA client and is typically used to interact with the IOTA Tangle, which is the underlying distributed ledger technology used by IOTA.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**//INSIDE OF THE SCRIPT**

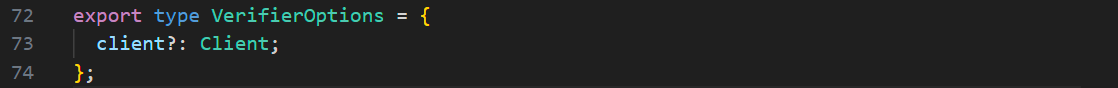
export type VerifierOptions = {

client?: Client;

};

The VerifierOptions type in the IOTA DID framework is used to specify the options for a verifier. It is defined as an object with an optional client property of type Client.

The client property allows you to specify a custom IOTA client to be used by the verifier. The IOTA client is responsible for interacting with the IOTA Tangle, which is the distributed ledger used by the IOTA network. By providing a custom client, you have the flexibility to use a different implementation or configuration for interacting with the Tangle.



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**