VON-X Documentation

Release 1.2.0rc1

Andrew Whitehead

Dec 24, 2018

Contents:

A connector for issuing credentials to TheOrgBook as part of the Verifiable Organizations Network (VON).

CHAPTER 1

vonx package

1.1 Subpackages

1.1.1 vonx.common package

Submodules

vonx.common.config module

Methods for loading and working with our standard YAML-based configuration files

vonx.common.config.**expand_string_variables**(*value*, *env: Mapping*, *warn: bool = True*)
Expand environment variables of form *$var* and *${var}* in a string

Parameters

- **value** (*str*) – The input value
- **env** (*Mapping*) – The dictionary of environment variables
- **warn** (*bool*) – Whether to warn on references to undefined variables

Returns The transformed string

vonx.common.config.**expand_tree_variables**(*tree*, *env: Mapping*, *warn: bool = True*)
Expand environment variables of form *$var* and *${var}* in a configuration tree. This is used to allow variable insertion in issuer and route definitions

Parameters

- **tree** – A sequence, mapping or other value
- **env** (*Mapping*) – The dictionary of environment variables
- **warn** (*bool*) – Whether to warn on references to undefined variables

Returns The transformed tree

vonx.common.config.**load_config**(*path: str*, *env=None*)
Load a YAML config file and replace variables from the environment

Parameters **path** (*str*) – The resource path in the form of *dir/file* or *package:dir/file*

Returns The configuration tree with variable references replaced, or *False* if the file is not found

vonx.common.config.**load_resource**(*path: str*) → TextIO
Open a resource file located in a python package or the local filesystem

Parameters **path** (*str*) – The resource path in the form of *dir/file* or *package:dir/file*

Returns A file-like object representing the resource

vonx.common.config.**load_settings**(*env=True*) → dict
Loads the application settings from several sources:

- settings.yml
- an optional application settings file defined by SETTINGS_PATH
- custom environment settings defined by ENVIRONMENT (ie. dev, prod)
- enviroment variable overrides

Parameters **env** – A dict of environment variables, or the value True to inherit the global environment

Returns A combined dictionary of setting values

vonx.common.config.**map_tree**(*tree*, *map_fn: Callable*)
Map one tree to another using a transformation function

Parameters

- **tree** – A sequence, mapping or other value
- **map_fn** (*Callable*) – The function to apply to each node, returing the new value

Returns The transformed tree

vonx.common.eventloop module

Methods and classes for working with asyncio event loops

class vonx.common.eventloop.**Runner**(*loop=None*)
Bases: object

Run a new event loop in a separate thread and allow tasks to be submitted to it

join()
Wait for the event loop thread to terminate

loop
Accessor for the event loop instance

run_in_executor(*executor: concurrent.futures._base.Executor*, *func: Callable*, **args*) → _asyncio.Future
Run a function in an executor, in the runner's event loop

Parameters

- **executor** – the Executor to use, may be None for the default ThreadPoolExecutor
- **func** – the function to run

- **args** – arguments to pass to the function

run_task(*coro: Awaitable*) → _asyncio.Future
Add a coroutine to the event loop, to be run at a later time

Parameters **coro** – the coroutine to be added

start(*wait: bool = True*) → None
Run the event loop in a new thread

Parameters **wait** – block until the event loop is running

stop(*wait: bool = True*) → None
Terminate the event loop thread

Parameters **wait** – block until the event loop has been stopped

vonx.common.eventloop.**run_coro**(*coro: Coroutine*)
Run an async coroutine and wait for the results

Parameters **coro** (*CoroutineType*) – The coroutine to execute

Returns The result of the coroutine

vonx.common.eventloop.**run_in_executor**(*executor: concurrent.futures._base.Executor*, *coro: Coroutine*) → concurrent.futures._base.Future
Run an async coroutine in an executor when we aren't already inside an event loop

Parameters **executor** – A `ThreadExecutor` or `ProcessExecutor` instance which will run the coroutine

Returns A *Future* which can be used to access the result of the coroutine

vonx.common.exchange module

Implementation of the shared Exchange message bus and related classes for sending and acting upon messages

class vonx.common.exchange.**Exchange**
Bases: object

A central message exchange hub for receiving requests and passing them to processors which may live in a different thread or process, but have a known identifier. Multiple processors may also respond to the same identifier in order to share processing. Responses are optional and can be tied to the original request.

is_registered(*to_pid: str*) → bool
Check if a listener is currently running

join() → None
Wait for the exchange to finish running

recv(*to_pid: str*, *blocking: bool = True*, *timeout=None*) → vonx.common.exchange.MessageWrapper
Receive a message from the bus

Parameters

- **to_pid** – The identifier of the recipient service
- **blocking** – Whether to sleep this thread until a message is received
- **timeout** – An optional timeout before aborting

Returns The next message in the queue, or None

register(*to_pid: str*) → bool
Register a listener on the exchange

send(*to_pid: str*, *wrapper: vonx.common.exchange.MessageWrapper*) → bool
Add a message to the bus, blocking until the processing thread is ready

Parameters

- **to_pid** – The identifier for the receiving service
- **wrapper** – The message to be added to the queue

Returns True if the message is successfully added to the queue

start(*process: bool = True*) → None
Start the message exchange as a thread or process

status() → dict
Retrieve the status from the polling thread

Returns int, 'processed': int, 'total': int} representing the total numbers of messages handled by the exchange

Return type A dict in the form {'pending'

stop(*drain: bool = True*) → None
Send a stop signal to the polling thread

class vonx.common.exchange.**ExchangeFail**(*value*, *exc_info=True*)
Bases: *vonx.common.exchange.ExchangeMessage*

An error class to represent an exception in message processing

This is not a subclass of `Exception` as that cannot be pickled and transported over the message bus

format() → str
Format this *ExchangeFail* instance as a string including the traceback, if any

class vonx.common.exchange.**ExchangeMessage**(**args*, ***kwargs*)
Bases: `object`

A common base class for exchange messages

get(*name: str*, *defval=None*)
Get a property of the message by name

Parameters

- **name** – the property name
- **defval** – the default value to return if the property is not defined

class vonx.common.exchange.**HelloProcessor**(*pid: str*, *exchange: vonx.common.exchange.Exchange*)
Bases: *vonx.common.exchange.MessageProcessor*

A simple request processor for testing response functionality or stress testing

class vonx.common.exchange.**MessageProcessor**(*pid: str*, *exchange: vonx.common.exchange.Exchange*)
Bases: `object`

A generic message processor which polls the exchange for messages sent to this endpoint and runs the abstract 'process' method to perform actions and send responses.

exchange
Accessor for the *Exchange* used by this request processor

get_message_target(*pid: str*) → vonx.common.exchange.MessageTarget
Quickly create a *MessageTarget* for a service on the same message bus

join() → None
Await our polling thread. *stop()* must be called in order to cause it to abort

pid
Accessor for the identifier of this request processor service

send(*to_pid: str*, *ident: str*, *message: vonx.common.exchange.ExchangeMessage*, *ref: str = None*, *from_pid: str = None*) → bool
Send a message to a recipient on the exchange

Parameters

- **to_pid** – The identifier of the recipient
- **ident** – The identifier of thie message, to be used by responses
- **message** – The content of the message
- **ref** – The identifier of the message being responded to
- **from_pid** – An optional override for the sender identifier

Returns True if the message was successfully added to the queue

send_noreply(*to_pid: str*, *message: vonx.common.exchange.ExchangeMessage*, *ref: str = None*, *from_pid: str = None*) → bool
Send a message with no reply expected

Returns True if the message was successfully added to the queue

send_stop_message() → bool
Send the service a stop signal to end processing

start(*_wait: bool = True*) → None
Run a thread to poll for received messages

stop(*wait: bool = True*) → None
Send a stop signal to the polling thread in order to abort polling

class vonx.common.exchange.**MessageTarget**(*pid: str*, *exchange: vonx.common.exchange.Exchange*, *from_pid: str = None*)
Bases: object

A wrapper for sending messages to a single target.

Example

```
>>> target = MessageTarget(target_pid, exchange, my_pid)
>>> target.send_noreply('hello')
True
```

exchange
Accessor for the *Exchange* used by this target

from_pid
Accessor for the identifier of the sending service

pid
Accessor for the identifier of the recipient service

send(*ident: str*, *message: vonx.common.exchange.ExchangeMessage*, *ref: str = None*, *from_pid: str = None*) → bool
Send a message to the recipient service

Parameters

- **ident** – The identifier used by the message response
- **message** – The message being sent
- **ref** – An optional identifier for the message being responded to
- **from_pid** – An optional override for the sender identifier

Returns True if the message was successfully added to the queue

send_noreply(*message: vonx.common.exchange.ExchangeMessage*, *ref: str = None*, *from_pid: str = None*) → bool
Send a message with no reply expected

Returns True if the message was successfully added to the queue

class vonx.common.exchange.**MessageWrapper**
Bases: tuple

A wrapper for a message being passed through the *Exchange* message bus

from_pid
str – The identifier of the sending service

ident
str – A unique identifier for the message, used to tag responses

message
ExchangeMessage – The message received

ref
str – An optional identifier for the message being responded to

from_pid
Alias for field number 0

ident
Alias for field number 1

message
Alias for field number 2

ref
Alias for field number 3

class vonx.common.exchange.**QueuedMessage**
Bases: tuple

A wrapper for a message queued to be sent to the exchange

to_pid
str – The identifier of the recipient service

message
ExchangeMessage – The message to be sent

message
Alias for field number 1

to_pid
Alias for field number 0

class vonx.common.exchange.**RequestExecutor**(*pid: str*, *exchange: vonx.common.exchange.Exchange*)
Bases: *vonx.common.exchange.MessageProcessor*

An subclass of *MessageProcessor* which starts a thread for each outgoing request to wait for responses. One of these should live in each process which wants to perform async requests via the *Exchange* (like a webserver process). It normally assumes that all incoming messages are simply responses to earlier requests. Processing should not block the main thread (much) to avoid breaking asyncio.

get_request_target(*pid: str*) → vonx.common.exchange.RequestTarget
Create a *RequestTarget* for a specific service

Parameters **pid** – the identifer of the target service

http
A quick accessor for a default HTTP client instance

http_client(**args*, ***kwargs*) → <MagicMock id='140274000677072'>
Construct an HTTP client using the shared connection pool

run_task(*proc: Awaitable*) → _asyncio.Future
Add a coroutine task to be performed by the runner

Parameters **proc** – the coroutine to be executed in the runner's event loop

run_thread(*proc: Callable*, **args*, *ident: str = None*) → _asyncio.Future
Add a task to be processed, as either a coroutine or function

Parameters

- **proc** – the function to be run in the `ThreadPoolExecutor`
- **args** – arguments to pass to the proc, if a function

runner() → vonx.common.eventloop.Runner
Accessor for the event loop runner instance used to execute tasks

start(*wait: bool = True*) → None
Initialize our `eventloop.Runner` and run our polling thread to listen for messages

start_process() → multiprocessing.context.Process
Start this executor in a new process

submit(*to_pid: str*, *request: vonx.common.exchange.ExchangeMessage*, *timeout: int = None*) → _asyncio.Future
Submit a message to another service and run a task to poll for the results

Parameters

- **to_pid** – the identifier of the target service
- **request** – the body of the message to be sent
- **timeout** – an optional timeout to wait before cancelling the request

tcp_connector
Return a connection pool associated with this event loop, which allows HTTP connection reuse

class vonx.common.exchange.**RequestTarget**(*executor: vonx.common.exchange.RequestExecutor*, *pid: str*)
Bases: `object`

An endpoint for a *RequestExecutor* which uses submit() to poll for responses to requests. It must be created within the same process as the executor instance

Example

```
>>> target = RequestTarget(executor, target_pid)
>>> target.request('hello')
Future<...>
```

executor
Accessor for the *RequestExecutor* instance

pid
Accessor for the target service identifier

request(*message: vonx.common.exchange.ExchangeMessage*, *timeout: int = None*) → _asyncio.Future
Send a request to the recipient service, awaiting the response in a method defined by the executor

Parameters

- **message** – The message to be sent
- **timeout** – An optional timeout for the message response

class vonx.common.exchange.**StopMessage**(**args*, ***kwargs*)
Bases: *vonx.common.exchange.ExchangeMessage*

Basic stop-processing message for *MessageProcessor* instances

class vonx.common.exchange.**ThreadedHelloProcessor**(*pid*, *exchange*, *blocking=False*, *max_workers=5*)
Bases: *vonx.common.exchange.HelloProcessor*

A threaded request processor for testing delayed, blocking and non-blocking responses

start(*_wait: bool = True*) → None
Run a thread to poll for received messages

start_process() → multiprocessing.context.Process
Start this demo processor as a process instead of a thread

vonx.common.exchange.**format_type_name**(*ctype*)
Convert a type or list of types to a string

vonx.common.manager module

Implemention of the generic *ServiceManager* class which is used to manage a collection of ServiceBase instances

class vonx.common.manager.**ConfigServiceManager**(*env: Mapping = None*, *pid: str = 'manager'*)
Bases: *vonx.common.manager.ServiceManager*

A *ServiceManager* subclass with standard configuration loading methods

config_root
Accessor for the value of the CONFIG_ROOT setting, defaulting to the current directory

load_config_path(*settings_key*, *default_path*, *env=None*) → dict
Load a YAML configuration file with defined variables replaced in the result

Parameters

- **settings_key** – the name of an environment variable defining an alternative configuration path
- **default_path** – the default path to the configuration file

Returns the parsed YAML configuration with variables replaced

services_config(*section: str*) → dict
Load a named section from the global services.yml configuration

Parameters **section** – the configuration key

class vonx.common.manager.**ServiceManager**(*env: Mapping = None*, *pid: str = 'manager'*)
Bases: *vonx.common.service.ServiceBase*

The standard *ServiceManager* class is responsible for starting the message exchange, registering itself as a service, starting any dependent services, and checking the status of those services. It should normally be run with *start_process()* before the web server process has forked.

add_service(*svc_id: str*, *service: vonx.common.service.ServiceBase*)
Add a service to the service manager instance

Parameters

- **svc_id** – the unique identifier for the service
- **service** – the service instance

env
Accessor for our local environment dict

exchange
Accessor for the Exchange this ServiceManager uses for messaging

executor
Return a per-process request executor which manages requests and polls for results coming from other services. Note: this is called for each worker process started by the webserver.

get_service(*name: str*)
Fetch a defined service by name

Parameters **name** – the string identifier for the service

Returns the service instance, or None if not found

get_service_message_target(*name: str*) → vonx.common.exchange.MessageTarget
Get an endpoint for one of the services defined by this manager. This Endpoint can be used for sending process-safe messages and receiving results.

Parameters

- **name** – the string identifier for the service
- **loop** – the current event loop, if any

get_service_request_target(*name: str*) → vonx.common.exchange.RequestTarget
Get an endpoint for sending messages to a service on the message exchange. Requests will be handled by the executor for this manager in this process.

Parameters

- **name** – the string identifier for the service
- **loop** – the current event loop, if any

get_service_status(*svc_id: str*) → dict
Fetch the status of a registered service

Parameters **svc_id** – the unique identifier for the service

proc_locals
Accessor for all process-local variables

Returns a dictionary of currently-defined variables

start(*wait: bool = True*) → None
Start the message processor and any other services

stop(*wait: bool = True*) → None
Stop the message processor and any other services

vonx.common.service module

Basic implementation and message classes for services running on the exchange

class vonx.common.service.**ServiceAck**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceResponse*

A simple acknowledgment response

class vonx.common.service.**ServiceBase**(*pid: str*, *exchange: vonx.common.exchange.Exchange*, *env: Mapping*)
Bases: *vonx.common.exchange.RequestExecutor*

The base class for services handled by the ServiceManager instance

send_stop_message() → bool
Send the service a stop signal to end processing

start(*wait: bool = True*) → None
Start the processing thread and any related services

class vonx.common.service.**ServiceFail**(*value*, *exc_info=True*)
Bases: *vonx.common.exchange.ExchangeFail*

A standard base class for errors returned from a service

class vonx.common.service.**ServiceRequest**(**args*, ***kwargs*)
Bases: *vonx.common.exchange.ExchangeMessage*

A standard base class for requests to a service

class vonx.common.service.**ServiceResponse**(**args*, ***kwargs*)
Bases: *vonx.common.exchange.ExchangeMessage*

A standard base class for responses from a service

class vonx.common.service.**ServiceStatus**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceResponse*

Request the status of a service

class vonx.common.service.**ServiceStatusReq**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceRequest*

Request the status of a service

class vonx.common.service.**ServiceStopReq**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceRequest*

Request a service to stop running

exception vonx.common.service.**ServiceSyncError**
Bases: Exception

An exception raised in response to a controlled failure during synchronization

class vonx.common.service.**ServiceSyncReq**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceRequest*

Request a service to perform a sync

vonx.common.util module

Utility functions and classes

class vonx.common.util.**JsonRepr**(*value*, *indent=2*)
Bases: object

Utility class to avoid JSON encoding debug output unless needed

class vonx.common.util.**MessageEncoder**(***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

Customize JSONEncoder to automatically encode ExchangeMessage instances

default(*o*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

class vonx.common.util.**Stats**(*logger=None*, *log_level=10*)
Bases: object

Measure combined statistics for various named tasks

class Timer(*stats*, *tasks*, *log_as=None*)
Bases: object

An instance of a timer that can be used in a with statement

end()
End the timer

start()
Start the timer

end(*handle*)
End a previously started set of tasks

results()

start(**tasks*, *log_as=None*)
Start a new set of tasks

timer(**tasks*, *log_as=None*)
Create a new timer for a set of tasks

vonx.common.util.**log_json**(*heading*, *data*, *logger=None*, *level=10*)
Utility method to log JSON data for debugging

vonx.common.util.**normalize_credential_ids**(*cred_ids*) → set
Clean up credential ID input

Module contents

vonx.common module initialization

1.1.2 vonx.config package

Module contents

A package containing standard configuration settings for von-x implementations

1.1.3 vonx.indy package

Submodules

vonx.indy.client module

vonx.indy.config module

vonx.indy.connection module

Base implemention of Connections, to be managed by `ConnectionCfg`.

class vonx.indy.connection.**ConnectionBase**(*agent_id: str*, *agent_type: str*, *agent_params: dict*, *conn_params: dict*)
Bases: `object`

Base interface for Connection implementations

close() → None
Shut down the connection

construct_proof(*request: vonx.indy.messages.ProofRequest*, *cred_ids: set = None*, *params: dict = None*) → vonx.indy.messages.ConstructedProof
Ask the target to construct a proof from a proof request

Parameters

- **request** – the prepared Indy proof request
- **params** – extra parameters for the API

generate_credential_request(*indy_offer: vonx.indy.messages.CredentialOffer*) → vonx.indy.messages.CredentialRequest

Ask the target to generate a credential request from our credential offer

Parameters

- **indy_offer** – the result of preparing a credential offer
- **creddef_id** – the ID of the credential definition

open(*service: IndyService*) → None

Initialize the connection

store_credential(*indy_cred: vonx.indy.messages.Credential*) → vonx.indy.messages.StoredCredential

Ask the target to store a credential

Parameters **indy_cred** – the result of preparing a credential from a credential request

store_credential_batch(*indy_creds: Sequence[vonx.indy.messages.Credential]*) → vonx.indy.messages.StoredCredentialBatch

Ask the target to store a credential

Parameters **indy_creds** – the prepared list of credentials

sync() → None

Perform any required synchronization

class vonx.indy.connection.**ConnectionType**

Bases: enum.Enum

Enumeration of supported connection types

HTTP = 'HTTP'

OrgBook = 'OrgBook'

holder = 'holder'

class vonx.indy.connection.**HolderConnection**(*agent_id: str*, *agent_type: str*, *agent_params: dict*, *conn_params: dict*)

Bases: *vonx.indy.connection.ConnectionBase*

ConnectionBase interface implementation for local holder services

close() → None

Shut down the connection

construct_proof(*request: vonx.indy.messages.ProofRequest*, *cred_ids: set = None*, *params: dict = None*) → vonx.indy.messages.ConstructedProof

Ask the target to construct a proof from a proof request

Parameters

- **request** – the prepared Indy proof request
- **params** – extra parameters for the API

generate_credential_request(*indy_offer: vonx.indy.messages.CredentialOffer*) → vonx.indy.messages.CredentialRequest

Ask the target to generate a credential request from our credential offer

Parameters **indy_offer** – the result of preparing a credential offer

open (*service: IndyService*) → None
Initialize the connection

store_credential (*indy_cred: vonx.indy.messages.Credential*) → vonx.indy.messages.StoredCredential
Ask the target to store a credential

Parameters **indy_cred** – the result of preparing a credential from a credential request

class vonx.indy.connection.**HttpConnection** (*agent_id: str*, *agent_type: str*, *agent_params: dict*, *conn_params: dict*)
Bases: *vonx.indy.connection.ConnectionBase*

A class for managing communication with an external agent over an HTTP connection

close () → None
Shut down the connection

construct_proof (*request: vonx.indy.messages.ProofRequest*, *cred_ids: set = None*, *params: dict = None*) → vonx.indy.messages.ConstructedProof
Ask the API to construct a proof from a proof request

Parameters **proof_request** – the prepared Indy proof request

generate_credential_request (*indy_offer: vonx.indy.messages.CredentialOffer*) → vonx.indy.messages.CredentialRequest
Ask the API to generate a credential request from our credential offer

Parameters **indy_offer** – the result of preparing a credential offer

get_api_url (*path: str = None*) → str
Construct the URL for an API request

Parameters **path** – an optional path to be appended to the URL

get_json (*path: str*)
A standard GET request to an API method

Parameters **path** – The relative path to the API method, including ? url parameters

Returns the decoded JSON response

open (*service: IndyService*) → None
Initialize the connection

path_prefix

post_json (*path: str*, *data*)
A standard POST request to an API method

Parameters

- **path** – The relative path to the API method
- **data** – The body of the request, to be converted to JSON

Returns the decoded JSON response

store_credential (*indy_cred: vonx.indy.messages.Credential*) → vonx.indy.messages.StoredCredential
Ask the API to store a credential

Parameters **indy_cred** – the result of preparing a credential from a credential request

store_credential_batch (*indy_creds: Sequence[vonx.indy.messages.Credential]*) → vonx.indy.messages.StoredCredentialBatch
Ask the API to store a list of credentials

Parameters **indy_creds** – the prepared list of credentials

class vonx.indy.connection.**HttpSession**(*method: str*, *http_client: <MagicMock id='140273997071248'> = None*, *timeout=None*)

Bases: object

Handle an exception or bad response from an HTTP request

check_status(*response: <MagicMock id='140273997138072'>*, *accept=(200, 201)*)

Check the HTTP status of a response to a previous request

client

Accessor for the ClientSession instance

vonx.indy.errors module

Common exception classes for Indy services

exception vonx.indy.errors.**IndyClientError**

Bases: *vonx.indy.errors.IndyError*

Base class for IndyClient-related errors

exception vonx.indy.errors.**IndyConfigError**

Bases: *vonx.indy.errors.IndyError*

Base class for IndyService errors arising from configuration issues

exception vonx.indy.errors.**IndyConnectionError**(*message: str*, *status=None*, *response=None*)

Bases: *vonx.indy.errors.IndyError*

A generic exception representing an issue with a ConnectionBase operation

exception vonx.indy.errors.**IndyError**

Bases: Exception

Base class for all exceptions thrown by IndyService and related classes

vonx.indy.manager module

vonx.indy.messages module

Message classes used to communicate with the IndyService

class vonx.indy.messages.**AgentStatus**(**args*, ***kwargs*)

Bases: *vonx.indy.messages.IndyServiceRep*

An agent status update

class vonx.indy.messages.**AgentStatusReq**(**args*, ***kwargs*)

Bases: *vonx.indy.messages.IndyServiceReq*

A request for an agent status update

class vonx.indy.messages.**ConnectionStatus**(**args*, ***kwargs*)

Bases: *vonx.indy.messages.IndyServiceRep*

A connection status update

class vonx.indy.messages.**ConnectionStatusReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request for a connection status update

class vonx.indy.messages.**ConstructProofReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to construct a proof from a proof request

class vonx.indy.messages.**ConstructedProof**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successfully constructed proof

class vonx.indy.messages.**Credential**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successful credential creation

class vonx.indy.messages.**CredentialDependencies**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The message class representing a credential's dependencies

class vonx.indy.messages.**CredentialDependenciesReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Get dependencies for a credential

class vonx.indy.messages.**CredentialOffer**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successful credential offer response :param data: the resulting credential offer :type data: dict

class vonx.indy.messages.**CredentialRequest**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successful credential request response :param cred_offer: the credential offer used as a basis :type cred_offer: CredentialOffer :param data: the resulting credential request :type data: str :param metadata: the credential request metadata :type metadata: dict

class vonx.indy.messages.**Endpoint**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The message class representing an agent's endpoint

class vonx.indy.messages.**EndpointReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Get endpoint for a did

class vonx.indy.messages.**FilterCredentialsReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Get credentials for an organization, matching a proof request dependencies

class vonx.indy.messages.**GenerateCredentialRequestReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to generate a credential request

class vonx.indy.messages.**GenerateProofRequestReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to generate a proof request

class vonx.indy.messages.**IndyServiceAck**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceAck*

A generic acknowledgement in response to an Indy service request

class vonx.indy.messages.**IndyServiceFail**(*value*, *exc_info=True*)
Bases: *vonx.common.service.ServiceFail*

For generic errors in processing Indy requests

class vonx.indy.messages.**IndyServiceRep**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceResponse*

A generic Indy service response base class

class vonx.indy.messages.**IndyServiceReq**(**args*, ***kwargs*)
Bases: *vonx.common.service.ServiceRequest*

A generic Indy service request base class

class vonx.indy.messages.**IssueCredentialBatchReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Issue a credential via a previously-registered connection

class vonx.indy.messages.**IssueCredentialReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Issue a credential via a previously-registered connection

class vonx.indy.messages.**LedgerStatus**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The response to a ledger status request

class vonx.indy.messages.**LedgerStatusReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to fetch the status of the remote ledger

class vonx.indy.messages.**OrganizationCredentials**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The message class representing an organization's credentials

class vonx.indy.messages.**OrganizationCredentialsReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

Get credentials for an organization

class vonx.indy.messages.**ProofRequest**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A message representing an Indy proof request

class vonx.indy.messages.**ProofSpecStatus**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The proof specification status update

class vonx.indy.messages.**RegisterAgentReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to register an agent

class vonx.indy.messages.**RegisterConnectionReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to register a connection

class vonx.indy.messages.**RegisterCredentialTypeReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to register a schema for publishing

class vonx.indy.messages.**RegisterProofSpecReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to register a proof request specification

class vonx.indy.messages.**RegisterWalletReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to register a wallet

class vonx.indy.messages.**RequestProofReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to get a proof from a connection

class vonx.indy.messages.**ResolveNymReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

The message class representing a request to resolve a DID

class vonx.indy.messages.**ResolveSchemaReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to resolve a schema which may be defined by one of our issuers

class vonx.indy.messages.**ResolvedNym**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The message class representing a DID resolution response

class vonx.indy.messages.**ResolvedSchema**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A request to resolve a schema which may be defined by one of our issuers

class vonx.indy.messages.**StoreCredentialReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request to store a new credential

class vonx.indy.messages.**StoredCredential**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successful response to storing a credential

class vonx.indy.messages.**StoredCredentialBatch**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A successful response to storing a credential batch

class vonx.indy.messages.**VerifiedProof**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

The message class representing a successful proof verification

class vonx.indy.messages.**VerifyProofReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

The message class representing a request to verify a proof

class vonx.indy.messages.**WalletStatus**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceRep*

A wallet status update

class vonx.indy.messages.**WalletStatusReq**(**args*, ***kwargs*)
Bases: *vonx.indy.messages.IndyServiceReq*

A request for a wallet status update

vonx.indy.service module

vonx.indy.tob module

Connection handling specific to using the OrgBook as a holder/prover

class vonx.indy.tob.**TobConnection**(*agent_id: str*, *agent_type: str*, *agent_params: dict*, *conn_params: dict*)
Bases: *vonx.indy.connection.HttpConnection*

A class for managing communication with the OrgBook API and performing the initial synchronization as an issuer

fetch_list(*path: str*) → dict
A standard request to a *list*-style API method

Parameters **path** – The relative path to the API method

path_prefix

sync() → None
Submit the issuer JSON definition to the OrgBook to register our service

vonx.indy.tob.**assemble_issuer_spec**(*config: dict*) → dict
Create the issuer JSON definition which will be submitted to the OrgBook

vonx.indy.tob.**encode_logo_image**(*config: dict*, *path_root: str*) → str
Encode logo image as base64 for transmission

vonx.indy.tob.**extract_translated**(*config: dict*, *field: str*, *defval=None*, *deflang: str = 'en'*)

Module contents

vonx.indy module initialization

1.1.4 vonx.web package

Submodules

vonx.web.helpers module

vonx.web.process module

vonx.web.render module

vonx.web.routes module

vonx.web.views module

Module contents

1.2 Module contents

vonx module initialization

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

G

H

I

J

L

M

N

O

P

Q

R

S

T

V

W